NM1 X.

# NASA CONTRACTOR REPORT

CR-121016

# SPACEBORNE COMPUTER MULTI-ELEMENT SYSTEM CONFIGURATION ARCHITECTURE REFINEMENT: TASK 1 REPORT

Prepared under Contract No. NAS8-26698 by

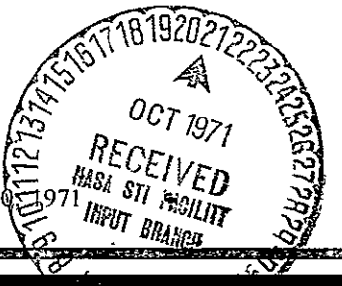

J.R. Kennedy, Sr.
R.T. Curran
B.P. Buckles
W.A. Hornfeck

FIELD SERVICES DIVISION
Aerospace Systems Center
Spaceborne Executive Project


For

NASA–GEORGE C. MARSHALL SPACE FLIGHT CENTER
Huntsville, Alabama

September 30, 1971







CSC
COMPUTER SCIENCES CORPORATION

TECHNICAL REPORT STANDARD TITLE PAGE

| 1. REPORT NO. | 2. GOVERNMENT ACCESSION NO. | 3. RECIPIENT'S CATALOG NO. |
|---|---|---|
| | | |

| 4. TITLE AND SUBTITLE | 5. REPORT DATE |
|---|---|
| Spaceborne Computer Multi-Element System Configuration Architecture Refinement: Task 1 Report | September 30, 1971 |
| | 6. PERFORMING ORGANIZATION CODE |
| 7. AUTHOR(S) J. R. Kennedy, R. T. Curran, B. P. Buckles, and W. A. Hornfeck | 8. PERFORMING ORGANIZATION REPORT # |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS<br>Computer Sciences Corporation<br>Field Services Div., Aerospace Systems Center<br>8300 South Whitesburg Drive<br>Huntsville, Alabama 35802 | 10. WORK UNIT NO. |
| | 11. CONTRACT OR GRANT NO.<br>NAS8-26698 |
| 12. SPONSORING AGENCY NAME AND ADDRESS<br>National Aeronautics and Space Administration<br>Washington, D. C. 20546 | 13. TYPE OF REPORT & PERIOD COVERED<br>Contractor Report |
| | 14. SPONSORING AGENCY CODE |

15. SUPPLEMENTARY NOTES
Work performed for George C. Marshall Space Flight Center Computation Laboratory.

16. ABSTRACT

This report presents an architectural study of a spaceborne computer system operating in a multi-element configuration. The system is described in prose and illustrated in functional flowcharts and diagrams. Requirements for high reliability and minimal SUMC logic impact are the major guidelines. The system comprises several SUMC central processor units, several input/output processors, a single system control unit, and several main memory units. The CPU architecture is described in terms of modifications to micro-instruction fields, main memory access, process control, input/output, configuration control, and scratch pad memory organization.

| 17. KEY WORDS | | 18. DISTRIBUTION STATEMENT |
|---|---|---|
| Computer Architecture | Reconfiguration | Unclassified - Unlimited |
| Multiprocessor | Multi-element Configuration | |
| Degraded Operation | Instruction Repertoire | |
| Onboard Computer | Microprogramming | |
| Spaceborne Computer | Central Processing Unit | |
| Space Ultrareliable Modular Computer | Memory Unit | |
| Spares Switching | Input/Output Processor | |
| Triple-Modular-Redundant | | |

| 19. SECURITY CLASSIF. (of this report) | 20. SECURITY CLASSIF. (of this page) | 21. NO. OF PAGES | 22. PRICE |
|---|---|---|---|
| Unclassified | Unclassified | 130 | |

MSFC - Form 3292 (May 1969)

# FOREWORD

The work reported herein was administered in the Systems Research Branch, Computer Systems Division, Computation Laboratory, MSFC, with Bobby C. Hodges assigned as Contracting Officer's Representative. In addition to his routine duties as Technical Monitor, Mr. Hodges has added significantly to our insight into and understanding of related NASA programs through careful planning, coordination with in-house effort, and encouragement.

Acknowledgement is due C. N. Swearingen and numerous MSFC Astrionics Laboratory personnel who originated the Space Ultrareliable Modular Computer design and have been the focal point of subsequent development. Their efforts have significantly advanced the state-of-the-art in flight computer hardware. Numerous discussions and technical interchanges with them have enhanced our understanding of the concepts of spaceborne computational hardware. Our appreciation also goes to personnel of the RCA Advanced Technical Laboratory who are currently fabricating a 16-bit CMOS design verification model of the SUMC, and have been willing to engage in frequent informal conversations that have greatly aided our research efforts.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

LIST OF TABLES

## DEFINITION OF SYMBOLS

| Symbol | Definition |
|---|---|
| ACK | Acknowledge |
| ALU | Arithmetic Logic Unit |
| AM | Action Map |
| ARG | Address Request Gating |
| ASD | Auxiliary Storage Devices |
| BA | Bank Address |
| BCC | Bus Channel Controller |
| BPA | Break Point Address |
| BPO | Break Point Operand |
| BRHA | Bank Register High Address |
| BRLA | Bank Register Low Address |
| $C_i$ | Data Bus Carrier Frequency |
| CLT | Control Logic and Timing |
| CPU | Central Processing Unit |
| CSS | Configuration Synthesis and Switching |
| CVTV | Concept Verification, Testing and Validation |
| DBT | Data Bus Terminal |
| DMA | Direct Memory Access |
| DR | Data Register |
| EASR | End Around Shift Register |
| ECO | Externally Controlled Output |
| FHD | Fixed Head Disk |
| FM | Format Memory |
| IAROM | Instruction Address Read Only Memory |
| II | IOP Input |
| IO | IOP Output |

DEFINITION OF SYMBOLS (Continued)

| Symbol | Definition |
|---|---|
| IOP | Input/Output Processor |
| LM | Local Memory |
| M | See MMU |
| MAR | Memory Address Register |
| MEC | Multi-Element Configuration |
| MI | Memory Input |
| MMU | Main Memory Unit |
| MO | Memory Output |
| MROM | Microprogrammed Read Only Memory |
| MRU | Multiplexer Register Unit |
| MUX | Multiplexer |
| PC | Program Counter |
| PCB | Program Control Block |
| PCO | Program Controlled Output |
| PRR | Product/Remainder Register |
| RDAU | Remote Data Acquisition Unit |
| $S_i$ | Switch Control Line |
| SCU | System Control Unit |
| SEQ-IC | Sequencer/Iteration Counter |
| SI | SCU Input |
| SM | Set-up Map |
| SO | SCU Output |
| SPM | Scratch Pad Memory |
| SUMC | Space Ultrareliable Modular Computer |
| TMR | Triple Modular Redundant |
| V | See VDSC |
| VDSC | Vote, Decision and Switch Control |
| WAR | Word Address Register |
| Z | Effective Address |

# SPACEBORNE COMPUTER MULTI-ELEMENT SYSTEM CONFIGURATION ARCHITECTURE REFINEMENT: TASK 1 REPORT

## SUMMARY

This report comprises an architectural study of a spaceborne computer system operating in a multi-element configuration (MEC). Sufficient detail is present to support the design of an on-board executive system. The study has been based upon computation requirements for extended space missions (little or no human maintenance) augmented by certain requirements based upon the Space Station, and upon the assumption that the SUMC (Space Ultrareliable Modular Computer) is used as the basic computing element. The most dominant study guideline was that architectural features should have minimal impact on the SUMC design.

The multi-element configuration is first discussed at the system level in order to provide an overview and the remainder of the report is then devoted to the individual element descriptions. The spaceborne computer multi-element system consists of several SUMC central processor units (CPUs), several input/output processors (IOPs), a single system control unit (SCU) and several main memory units. The interconnection of these elements by appropriate system buses can be accomplished under program control, thus achieving a dynamically reconfigurable system. Provided that a sufficient number of processors are available, the system could operate in a multiprocessing mode, TMR (triple-modular-redundant) mode, dedicated simplex mode or combinations of these.

Main memory for the system will consist of a number of identical 8K x 36 bit memory units. System organization allows any processor (CPU or IOP) currently operating to access up to 32 main memory units.

The major element of the MEC is the central processor. Since the system structure as depicted in this report is based upon the SUMC, the approach to CPU architectural specification is to summarize the baseline SUMC definition, and then define departures from this baseline. These departures are shown to be necessary and sufficient for efficient operation in a multiprocessor environment.

The CPU architecture needed to achieve efficient multiprocessor operation is described in terms of modifications to microinstruction fields, main memory access, process control, input/output, configuration control, and scratch pad memory organization. Additional special instructions are also discussed which are either required for MEC operation or desirable for additional programming effectiveness.

The single system control unit acts as a system supervisor and the functions it performs are principally supportive. The SCU could be implemented as a simplex SUMC unit operating as an internally redundant system controller. The role of the SCU during configuration control, CPU control, and process control is discussed, and the SCU architecture is defined.

The input/output processors provide the logical interface between the other elements of the MEC and the variety of peripheral devices that can be connected to a digital data bus through data bus terminals such as those baselined for the space station. Each IOP could also be implemented as a basic SUMC unit having the capability to control data transfers, monitor bus operations, and communicate with system CPUs. The IOP would then free the CPUs from many of the procedures involving data transfers and I/O operations.

# SECTION I. INTRODUCTION

This report is submitted in compliance with requirements of NASA Contract Number NAS8-26698 for an interim report of a spaceborne computer system operating in a multi-element configuration. The current study is directed toward architectural refinements with subsequent work to be devoted to design of the software executive.

The computational requirements of an extended space flight mission such as the Space Station/Base necessitate a processing system of considerable adaptability. Failure tolerance, power consumption, and throughput represent parameters which frequently change in value during the mission life-span. Research efforts directed toward achieving this flexibility have resulted in the design (and current fabrication) of the Space Ultrareliable Modular Computer (SUMC) by the Marshall Space Flight Center Astrionics Laboratory. Supporting elements and subsystems, at various levels of detail, have been proposed. It is the purpose of this report to expand the definition of these elements and to describe their inter-relationships sufficiently to support the development of a detailed on-board executive system design.

This effort was divided into a basic cycle of two steps. First, for each elemental system component an element description and functional design (if available) were chosen from previous research to represent the baseline approach. This baseline was used to establish a framework for discussion and to derive minimum capability criteria. Second, modifications and additions to the functional design were incorporated to support the inter-element communications necessary for performance of basic processing functions as well as reconfiguration and spares switching. Where necessary, supplemental detail was included to elucidate or demonstrate feasibility of the derived approach.

For one element, the system control unit (SCU), a deviation from the above pattern occurs. Available literature is characterized by a lack of detail explicitly describing configuration control mechanisms. In an effort to propound a viable, coherent approach, an SCU, fabricated from SUMC logic, is included as part of the processing system configuration to fill this void.

The initial portion of this report delineates the gross relationships of the SCU and other elements to the total system. The remainder is devoted to analysis of the specific organization of each key element of the configuration. Emphasis is placed on the main memory units (MMUs), central processing units (CPUs), SCU, and I/O processors (IOPs). It was not necessary to devote equal attention to the remote data acquisition units (RDAUs) and peripheral devices since the structure of the IOP is sufficiently flexible to negate their impact on system design.

# SECTION II. MULTI-ELEMENT CONFIGURATION (MEC) OVERVIEW

This section provides an overview of the major components comprising the multi-element configuration (MEC). The purpose of the overview is to discuss in system-level terminology the functional nature of the various system elements and their gross interrelations. This overview provides a framework for functional specifications given in Section III.

A summary of the MEC is given first. This summary is based upon a generic diagram of element interconnections. Buses for inter-element data and control exchange are discussed, and a scheme for switching is presented. Based upon this scheme, reconfiguration and spares switching is summarized and several examples of feasible configurations are presented.

## A. Configuration Summary

Figure 1 illustrates the MEC in a generic form. With the exception of the blocks labeled "$V_{XX}$," "LM," and "SCU," each block represents one or more copies of the symbolized element. For instance, the block labeled "M" represents one or more electronically equivalent main memory units (MMUs), each having several identical sets of input lines (one set for each processor potentially having access to the main memory unit) and corresponding sets of output lines. The pairs (input and output) of corresponding processor access lines are referred to as processor access "ports." Thus there is a distinct port for each of several processors that may have access.

The input port set is comprised of both memory input data lines and processor-to-memory control lines; the output port set is similarly comprised of memory output data lines and memory-to-processor response (or acknowledge) lines. The main memory unit contains sufficient logic for selection of one and only one port to establish a communications path to one and only one processor during a small time interval. By appropriate logic, the selection criteria can be organized in a number of ways. Preferential logic is usually employed to favor input/output processors over central processors in the event that two such processors simultaneously request access.

Except for this possible preferential selection, the main memory unit functions and responds identically in communications with central processing (labeled "SUMC") and input/output processing (labeled "IOP") units of the MEC diagram. For this reason, the access ports for SUMCs and IOPs are indistinguishable.

The Space Ultrareliable Modular Computer (SUMC) elements function as the system central processor units (CPUs). The CPUs are (essentially)

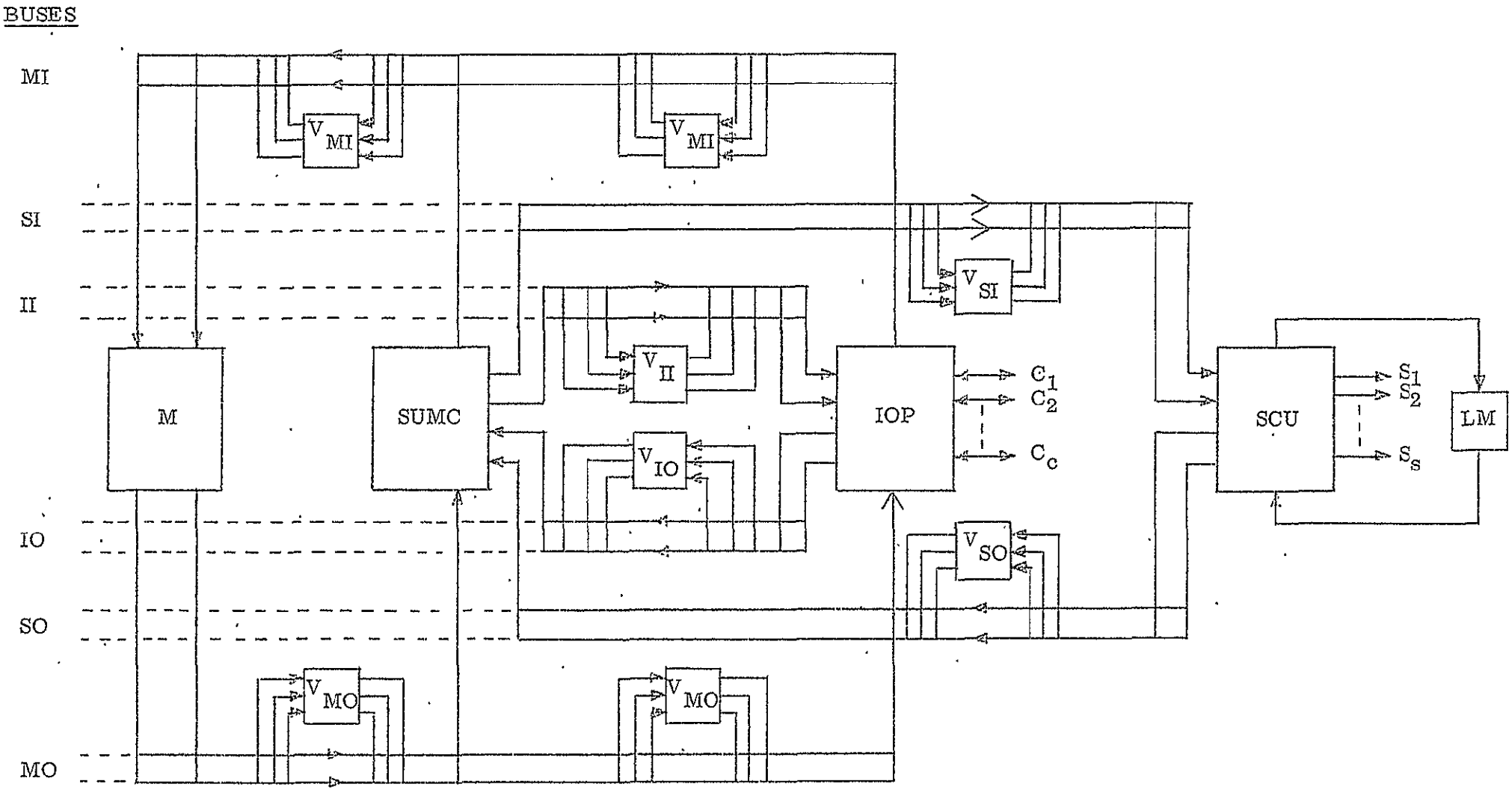



FIGURE 1

MULTI-ELEMENT CONFIGURATION

unaltered SUMCs having a single set of main memory access lines connectable to one of several main memory access buses (MI and MO in figure 1). Connection of one or more main memory elements to this bus structure through a corresponding port thus provides the necessary path for CPU access to those main memory elements.

In addition to the main memory access lines, a set of I/O lines for control of one or more IOPs is provided. These lines (II and IO in figure 1) correspond to the 18 high-numbered PRR bits (18-35) for CPU control and data output to the IOPs, and the 18 high-numbered MPXB1 bits (18-35) for IOP status, requests, and data input to the CPU. All IOPs controllable by a particular CPU are accessible through a unique port functionally similar to the main memory ports. Each IOP on the I/O lines of a CPU is uniquely addressable thus permitting a CPU-initiated dialog. An IOP-initiated dialog is supported through an IOP-to-CPU "poll request" (interrupt) control line signal followed by a CPU-controlled poll of all connected IOPs until an acknowledge from the requesting IOP is recognized. When an acknowledge is received, the CPU I/O lines are set to indicate "busy" until the dialog is completed. The poll request (or I/O interrupt) line is switched between the control logic and timing block and MPXB1 in the SUMC.

The I/O lines are used primarily to initiate IOP action and to check or sense status. Consequently, this traffic is low. The pair of unidirectional buses should therefore be adequate to support multiple IOPs. Transfer of a limited volume of device/CPU data (say 15 characters/second for pluggable computer system console operations) could also be sustained with little or no system degradation.

The IOP has multiple access ports for control by several CPUs. In addition, each IOP is connectable through one of several buses for access to main memory elements having a port connected to the IOP's memory access bus.

The "Cs" shown on the IOP block are representative of data bus carrier frequencies corresponding to several "channels." The IOP contains modem pairs for each channel frequency. A bit-serial keyed amplitude modulation scheme is representative of the capability envisioned.

The IOP is depicted in subsequent text as a modified SUMC microprogrammed to perform main memory program controlled input/output. The main memory program is comprised of a set of specially formatted IOP commands (instructions) structured to direct the IOP in the transfer of I/O data between main memory elements and data bus terminals, and in the initiation and control of data transfers between arbitrary devices attached to the

data bus (assuming the devices have this capability). As is discussed in more detail later, hardware additions to the basic SUMC to transform it into an IOP for support of input/output include a channel selection input multiplexer, a multiplexer for CPU input line selection, selectors (demultiplexers) for both CPU and channel output, and appropriate control logic and timing to efficiently support the input/output function.

The depicted IOP is a combined "input/output processor" and "bus control unit." Because of the SUMC stored logic control capability, and the inherent ability to monitor data bus activity via the added channel demodulators, a flexible device with high growth potential can be fabricated using SUMC chips. This approach seems to offer a cost effective method for achieving applicability to a broad spectrum of missions.

The system control unit (SCU) block is representative of a functionally simplex unit having system-level executive control over configuration switching actions, CPU dispatching (allocation of CPU time to programmed processes), and redundant mode operations failure detection, isolation, and spares switching.

The SCU is envisioned to operate (always) as an internally redundant (say, TMR with spares) system controller. It maintains a map of the current configuration and actuates switching networks to accomplish reconfiguration and spares switching under the direction of executive routine control. It also performs the dispatching function on the basis of a process ready list. Special instructions are defined for execution on the CPUs. These instructions result in requests made by the CPUs of the SCU.

The SCU has an access port for each CPU consisting of the low-numbered 18 SUMC PRR bits for CPU-to-SCU transfers, and the low-numbered 18 SUMC MPXB1 I/O data input lines to AD1 for SCU-to-CPU transfers. Thus, the 36 bit SUMC I/O data paths are shared by the SCU (high-numbered 18) and IOPs (low-numbered 18). An additional "attention" line from the SCU to each CPU is required. The data and command transfer volume between CPUs and the SCU is low, being primarily of a control nature. For this reason, 18 bits for each direction is felt to be adequate.

The "Ss" shown on the SCU block of figure 1 are switch control lines for the purpose of connecting main memory, central processor, and input/output processor element plug positions to the various system buses to establish the communication paths necessary for a given system structure.

The SCU is envisioned to operate under program control out of the small local memory block labeled "LM." LM is internally redundant in a manner consistent with SCU redundancy. It is estimated that LM will be

about 8000 16 bit words, and that the SCU work load is sufficiently low to suggest an implementation based upon a 16 bit version of the SUMC with a small instruction repertoire. The SCU is tentatively shown having no access to system main memory since system operation can be effected without SCU access to main memory; but, since the location of SCU programs is somewhat arbitrary, final configuration selection is temporarily left open for further analysis.

The remaining blocks in the diagram labeled "$V_{xx}$" are voting and disagree detection logic to support redundant configurations only. The blocks are shown having three sets of lines to support a TMR configuration. The label subscript "xx" has the following meaning:

| xx | Meaning |
|---|---|
| MI | MMU input bus |
| MO | MMU output bus |
| II | IOP input bus |
| IO | IOP output bus |
| SI | SCU input bus |
| SO | SCU output bus |

$V_{ii}$ and $V_{si}$ are identical, as are $V_{io}$ and $V_{so}$, because the number of lines involved is identical (and, of course, the functions are identical). Thus, four distinct voting and disagree detection networks are required, differing only in data path width.

Not shown are identical lines from each $V_{xx}$ going into the SCU for the purpose of indicating failures and identifying the disagreeing (TMR) path. (Reference 1 contains a discussion of the concepts involved in failure detection, configuration control, and switching that is the basis for the MEC scheme discussed here.)

In order to establish realistic numbers to be used for the development of tables, instruction fields, etc., a complement of elements comprising the MEC is assumed as follows:

---

[1] Kennedy, Sr., J. R.: SUMC Multiprocessor Configuration Control Analysis and Specification. Contractor Report Prepared under NASA Contract NAS8-18405 by Computer Sciences Corporation, Huntsville, Alabama, June 14, 1971.

Number of Elements Assumed:

| | | |
|---|---|---|
| CPU | – | 8 } spares included |
| IOP | – | 4 } |
| MMU | – | 32 } |
| SCU | – | 1 + possible spares |
| VMI | – | 2 + 6 possible spares |
| VMO | – | 2 + 6 possible spares |
| VII | – | 1 + 7 possible spares |
| VIO | – | 1 + 7 possible spares |
| VSI | – | 1 + 7 possible spares |
| VSO | – | 1 + 7 possible spares |

B. System Buses

Data flow is accommodated between subsystems over six sets of buses comprising

- Main memory access buses,
- Input/output processor buses, and
- System control unit/SUMC buses.

Main memory access buses provide for the data paths and control signals required by the SUMCs and the IOPs to store and retrieve information from the main memory elements. MI is comprised of 32 bits data, 18 bits of address information, 5 bits of control information, and 7 bits for parity, giving a total of 62. Thirty-two bits for data transfer, 4 control bits, and an additional 4 bits for parity, gives a total width of 40 bits for MO. Buses are required for each SUMC and each IOP resulting in an assumed total of 12 MI and MO buses.

In addition to main memory communication buses, there are two other sets of buses: the input/output processor buses and system control unit buses which provide for communication capability between the following system elements:

- SUMC/Input Output Processor (IOP), and
- SUMC/System Control Unit (SCU), respectively.

Input lines required for the IOP and SCU total 18. On the output side the IOP and SCU have 19 lines (18 with parity, 1 control). Table 1 summarizes the system bus structure.

TABLE 1. COMPUTER SYSTEM BUS COMPLEMENT

| | Bus | No. Lines | Minimum No. Buses | Remarks |
|---|---|---|---|---|
| Memory Input | (MI) | 62 | 12 | One for each SUMC and IOP |
| Memory Output | (MO) | 39 | 12 | One for each SUMC and IOP |
| SCU Input | (SI) | 18 | 8 | One each SUMC |
| SCU Output | (SO) | 19 | 8 | One each SUMC |
| IOP Input | (II) | 18 | 8 | One each SUMC |
| IOP Output | (IO) | 19 | 8 | One each SUMC |

C. Configuration Synthesis and Switching (CSS)

As mentioned previously in the summary, the MEC is capable of assuming many different configurations. Since the structure depicted is general with regard to data path organization, it is possible to operate several configurations simultaneously. These configurations can be similar or not, or functionally dedicated or not, depending on the mission requirements for reliability, allowable power consumption, throughput, and other identifiable parameters that can in some arbitrary way be associated with a specific configuration.

The potential for variability in configuration is limited at any given time primarily by the number of serviceable system elements of each type, and the number of usable data paths that can be established. An additional constraint on the variability in configuration is, of course, the existence of one or more programmed processes for control and allocation of the elements comprising the various configurations. Since the purpose of subsequent tasks is to analyse and define these programmed processes, this report defines a capability for attaining a high degree of configuration variability, and assumes that programmed processes can be defined to effectively utilize the capability.

With regard to the CSS function, this report discusses refinements to the concepts outlined in reference 1. The refinements consist mainly of organizing a set of rather general instructions into a compatible 32 bit word instruction format based upon an assumed number of available system elements of various types. Additional refinements are comprised of formatting a system map that conforms to the assumed element set, and a division of functional responsibility between executive routine algorithms executing on a CPU and executive routine algorithms executing on the SCU. This division of responsibility unambiguously delineates the CPU/SCU communications dialog required to accomplish the CSS function.

The role of the CPU executive is to construct a specific configuration map based upon element and bus availability and status. Availability and status information is obtained by a CPU from the SCU which maintains a current map of connections, and element and bus status. This availability and status information is returned to the CPU executive in the form of sense instruction responses. Once a map has been constructed by the CPU executive, it is transferred to the SCU. The executive routine algorithms in the SCU use this map to construct the necessary switch network commands for achieving the desired system structure. After all switching operations have been carried out, the SCU forces all active (switched online) CPUs to fetch their next instruction from a CPU executive specified main memory location. This action completes the transformation from executive control of one system structure to executive control of another system structure. Subsequent transformations are accomplished in the same manner.

At least three methods for achieving a specific structure are supported by the scheme outlined. The first is a programmed algorithmic method involving dynamic inventory of system resources and program controlled selection on the basis of availability. This scheme dynamically constructs a system map under program control on the basis of program structure. The second scheme is a prestored or externally constructed method wherein a specific configuration map is supplied to the CPU executive. The executive will take the necessary action to achieve the supplied configuration.

The third, and perhaps most interesting, is based upon a combination of the first two where the program structure is a form of programmed minimization of a cost function based on several parameters to determine an optimal structure. In this method a set of prestored system maps, each having an associated precalculated measure of reliability, power consumption, throughput, etc., would be used to dynamically minimize the selected cost function. There could be different cost functions for each mission but, more importantly, the applicable cost function could vary within a mission – perhaps on the basis of phase. Many variations on the last scheme are, of course, possible. In summary, these three schemes with variations are available for specifying and achieving configuration control:

- Program structured,
- System Map structured, and
- Parameterized Optimally structured..

## D. Configuration Examples

Several examples of specific configurations based on the generic diagram of figure 1 are given for completeness to serve as a basis for illustrating principles, and to develop structure-related definitions.

The principal property displayed by these different organizations is that of "structure." A system structure can be specified by its "class," "degree," "association," and "configuration," as follows:

1. Class Specifier:

   Uniform
   Non-Uniform

2. Degree Specifier:

   Maximum
   Full
   Minimum
   Partial

3. Association Specifier:

   Dedicated
   Non-Dedicated

4. Configuration Specifier:

   Simplex
   Multiple Simplex
   Redundant
   Multiprocessor
   Multisystem

These terms are defined as follows:

- Uniform - all bus switch settings are such that bus and port addresses are linearly related, and there is symmetry in the switch settings for input and output buses in a corresponding bus pair.

- Maximum - the largest subset of the total structure, spares included, that can be logically operated on-line (if elements have failed, a maximum degree may not be attainable).

- Full - all operable elements that can be logically operated on-line are connected (no greater throughput can be obtained without a change in class, association, and/or configuration).

- Minimum - the smallest logically operable subset of a total structure (the entire structure is inoperable if a minimum degree cannot be attained for some class, association, and configuration combination).

- Partial - between minimum and full.

- Dedicated - some or all of a structure is associated with some programmed function to the possible exclusion of other programmed functions.

- Simplex - A single CPU system having no multiple CPU expansion capability short of reconfiguration.

- Multiple Simplex - several simplex configurations with each having no resources allocation capability outside the domain of its own simplex configuration (for instance, no shared memory; this configuration is dedicated).

- Redundant - a configuration that is functionally simplex but is comprised of multiple elements of each type performing the same functions for the purpose of comparison to (at least) detect errors.

- Multiprocessor - a configuration having multiple CPUs and some provision for programmable shared storage or some other form of programmable interprocessor communication.

- Multisystem - a structure configuration comprised of a combination of configurations.

Figures 2 and 3 provide examples of two structures illustrating the flexibility of figure 1 and the specifiers defined above.

M - MAIN MEMORY
C - CENTRAL PROCESSOR
I - INPUT/OUTPUT PROCESSOR
S - SYSTEM CONTROL UNIT
L - LOCAL MEMORY

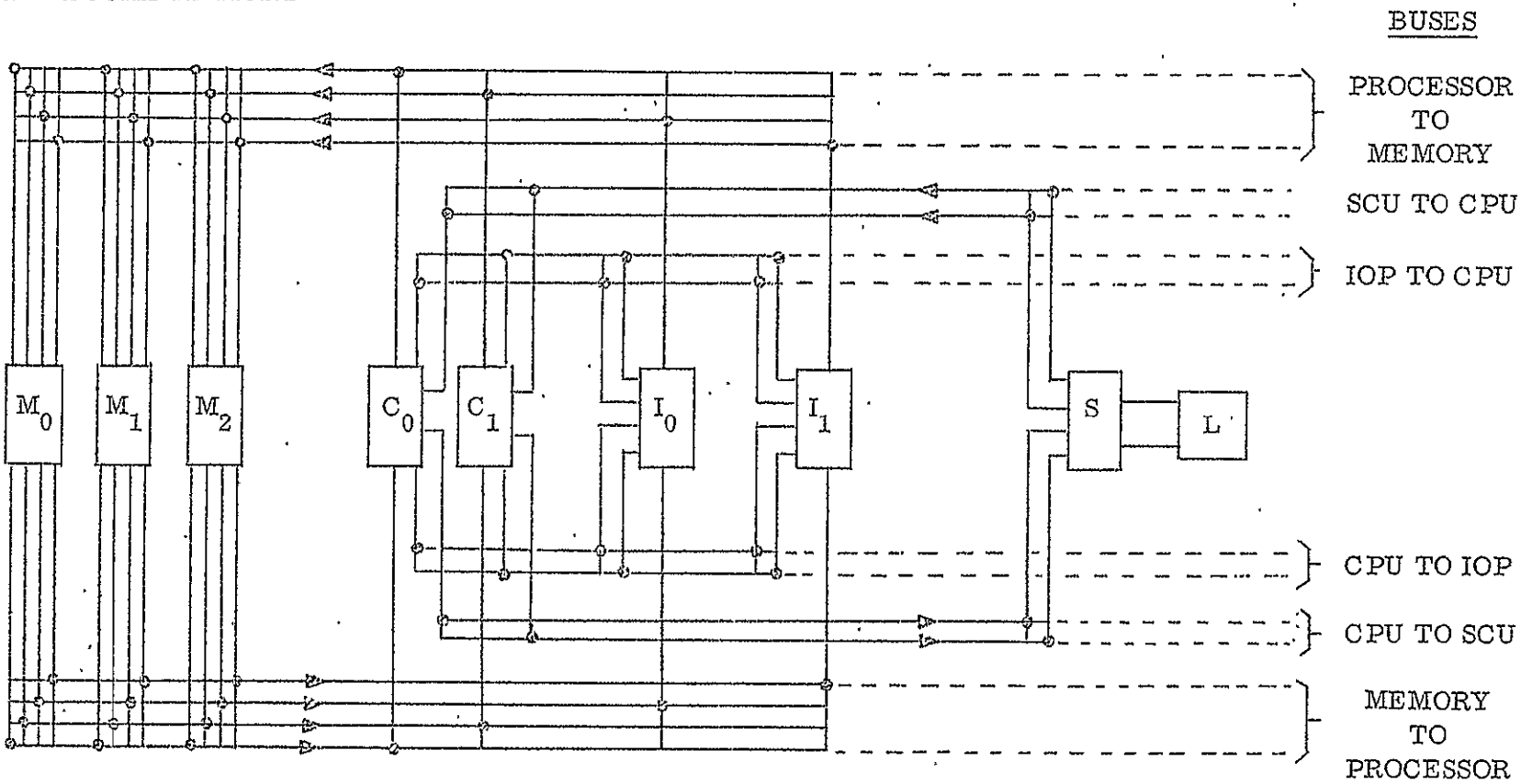



FIGURE 2

UNIFORM FULL NON-DEDICATED MULTIPROCESSOR STRUCTURE

M - MAIN MEMORY
C - CENTRAL PROCESSOR
I - INPUT/OUTPUT PROCESSOR
S - SYSTEM CONTROL UNIT
L - LOCAL MEMORY

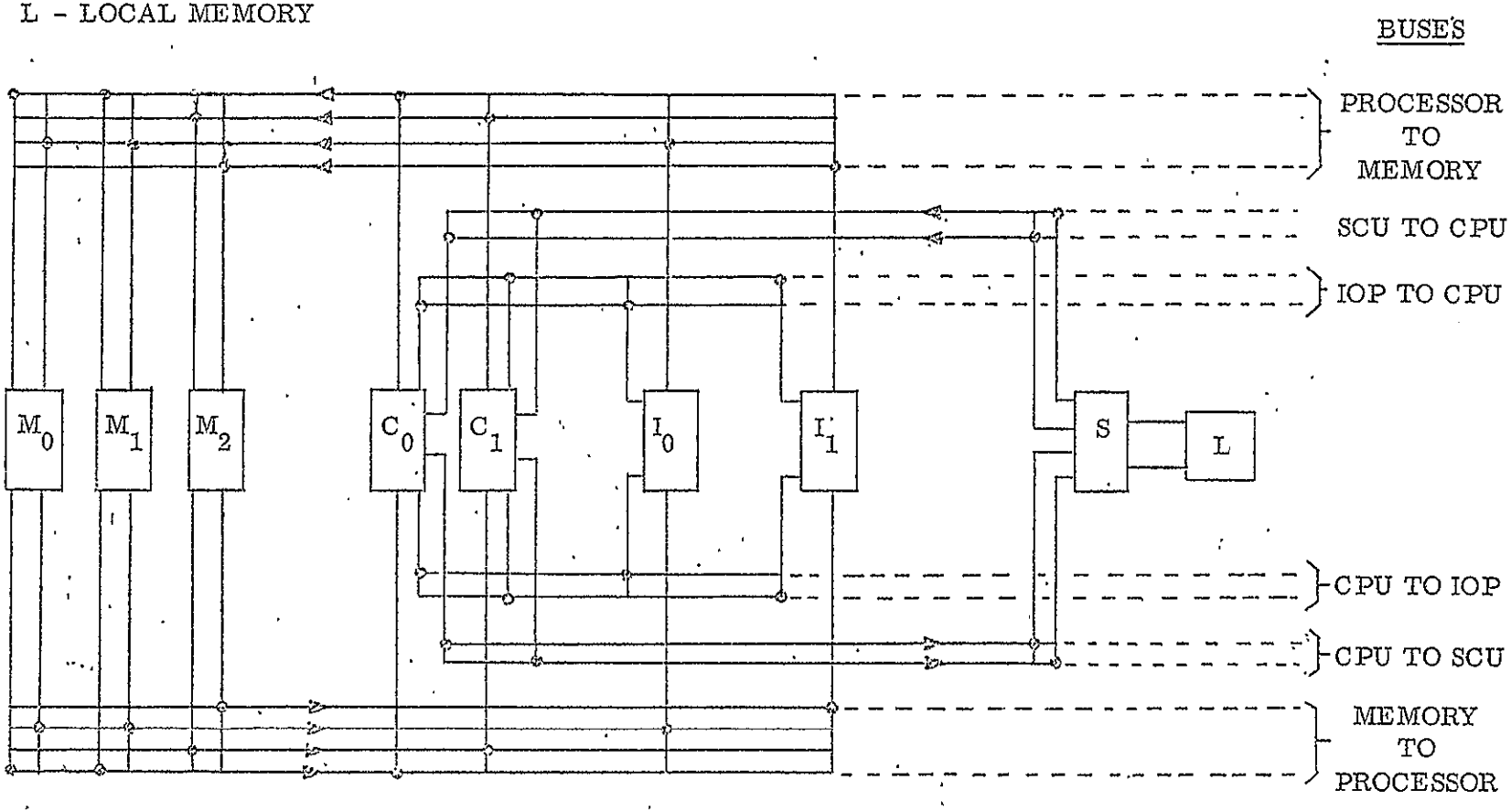



FIGURE 3

UNIFORM FULL DEDICATED MULTIPLE SIMPLEX STRUCTURE

## SECTION III. ELEMENT DESCRIPTIONS

This section provides a detailed level analysis of the major components comprising the multi-element configuration. Particular attention is devoted to the methodology and content of inter-element communication and internal element functions supporting this function. The confluence of these two areas has a major impact on the optimization (size) and efficiency (operating speed) of the on-board executive system.

Main memory units are discussed first, followed by the central processing unit. Considerable functional support detail is included for the CPU, including recommended special instructions, since it is the focal point of most system functions. The SCU and IOPs are discussed along with their role in the system dialog.

### A. Main Memory Units (MMUs)

A baseline memory organization is given in reference 2, described as the basic operating memory (BOM). The discussion below does not alter the derived BOM concepts of multi-port access to 8K memory modules.

1. Organization. Main memory is distributed among identical units, each 8K x 36 bits, modularly expandable to 32 units. Figure 4 depicts processor access gating to a single memory module and figure 5 details a generic memory module.

a. Processor access control. Figure 1 illustrates two unidirectional ports connecting each processor (CPU or IOP) to each MMU. Input ports consist of:

- 18 address lines,
- 32 data lines,
- 4 control lines (plus an access request line), and
- 7 parity lines for data and address validation.

---

[2] Eastin, Earl: Shuttle Computation System. Contractor Report SP-233-0252 prepared for MSFC by Sperry Rand Corporation under NASA Contract NAS8-20055, Huntsville, Alabama, June 8, 1970.

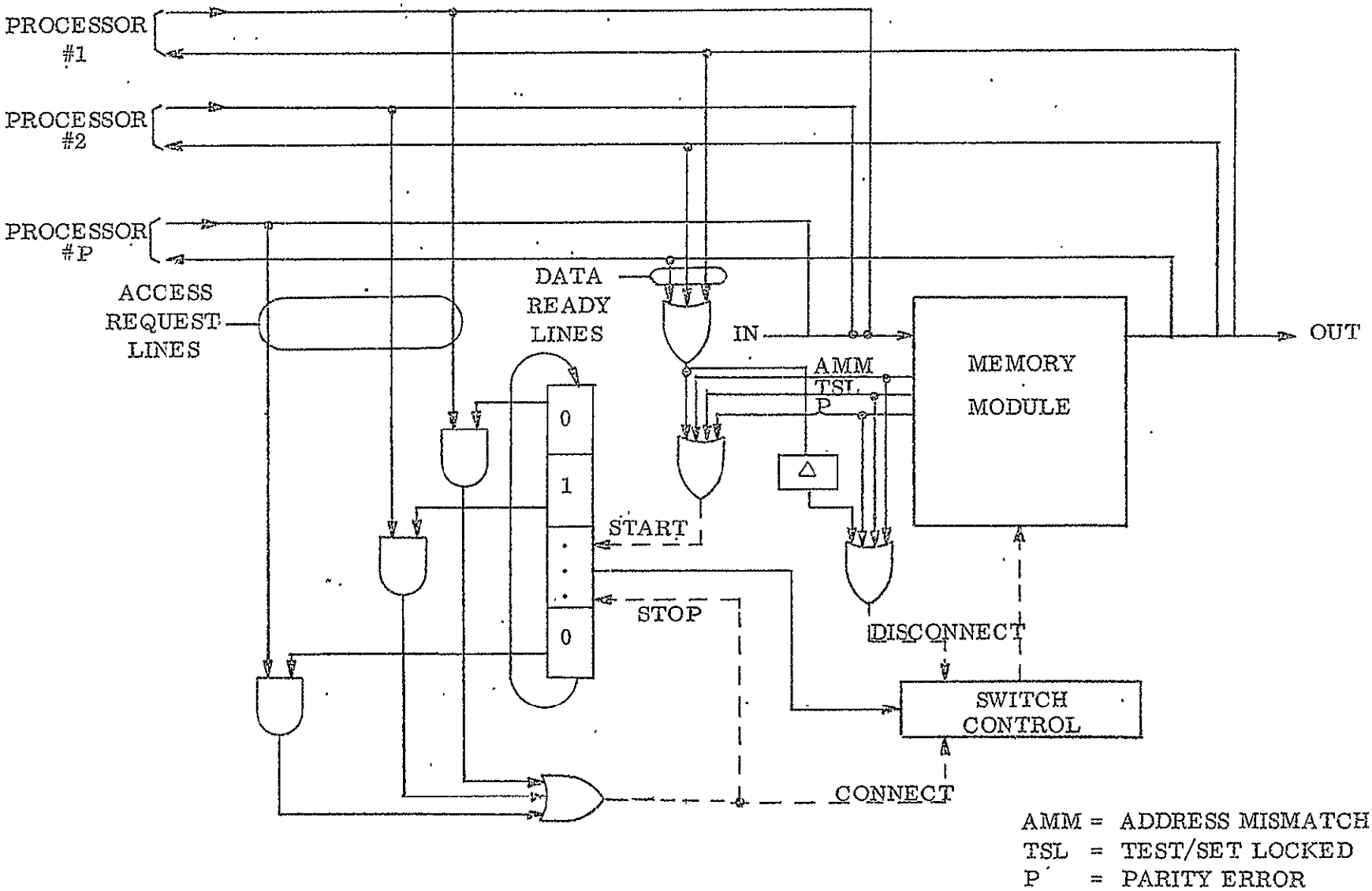



FIGURE 4

MEMORY ELEMENT - PROCESSOR ACCESS CONTROL

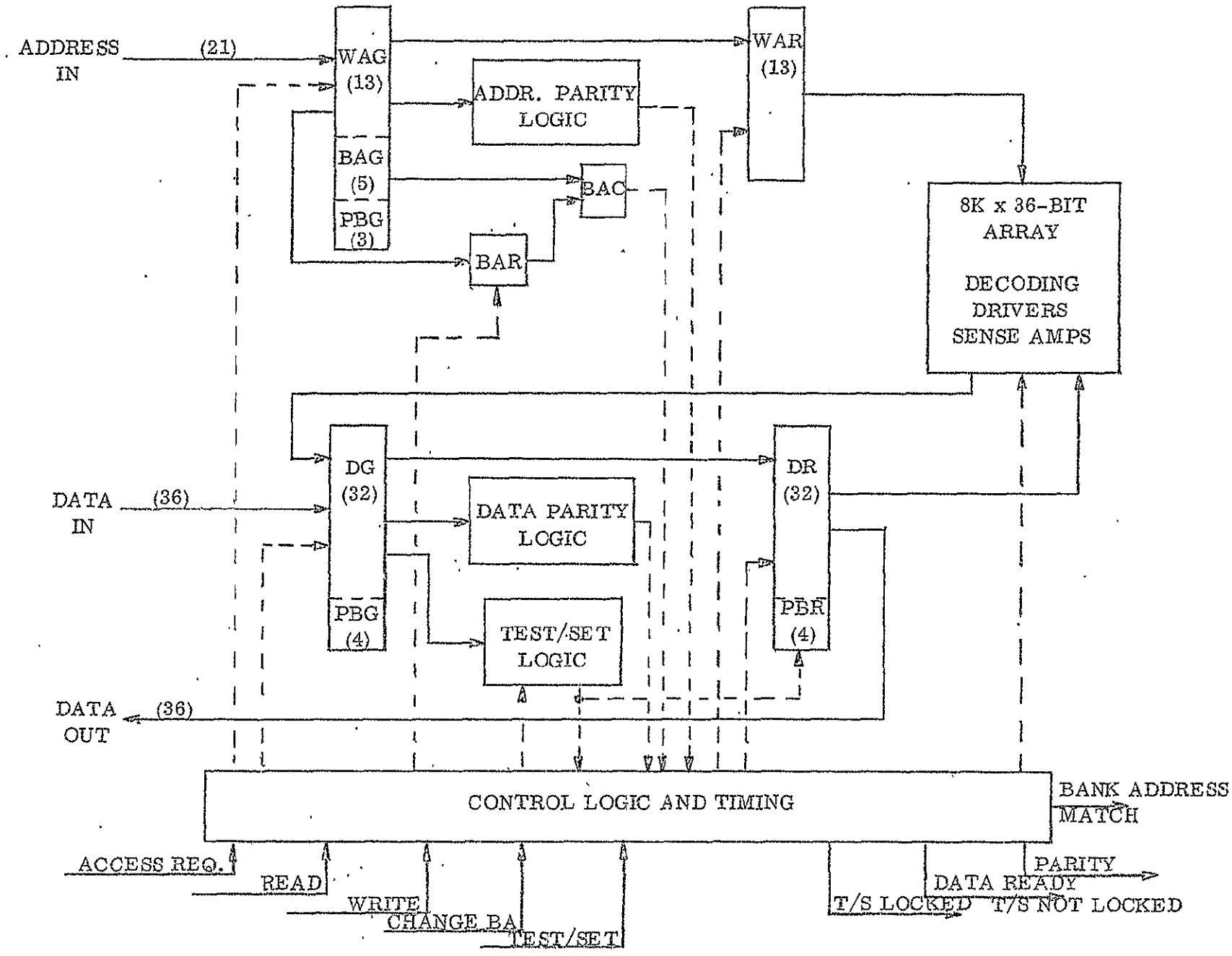



FIGURE 5

MULTIPROCESSOR MAIN MEMORY MODULE

TABLE 2. MEMORY MODULE LEGEND

| | |
|---|---|
| WAG | Word Address Gating |
| BAG | Bank Address Gating |
| PBG | Parity Bits Gating |
| BAR | Bank Address Register |
| BAC | Bank Address Comparator |
| WAR | Word Address Register |
| DG | Data Gating |
| DR | Data Register |
| PBR | Parity Bits Register |
| T/S | Test/Set |

Output ports consist of:

- 32 data lines,
- 4 control lines, and
- 4 parity lines for data validation.

Memory element/processor access control is accomplished as shown in figure 4. Access request gates continuously monitor the access request control lines of connected processors. An End Around Shift Register (with one bit set) sequentially scans for a request and signals the Switch Control when a processor access request is recognized. The switch control is capable of connecting and disconnecting the input and output ports from any processor to the memory module.

b. Main memory module. Figure 5 and its associated legend (table 2) depict the logical elements required internal to each module. In addition to an 8K x 36 bit (32 data, 4 parity) storage array with address decoding and sense logic, the functional units are:

- Control Logic and Timing for sequencing and synchronization of internal events;
- Word Address Gate (13 bits), Bank Address Gate (5 bits), and Parity Bits Gate (3 bits) for the routing of information from the address lines;
- Address Parity Logic for address parity validation;
- Bank Address Register (5 bits) for storage of access key of memory module;
- Bank Address Comparator for comparing Bank Address Register and Bank Address Gate contents;
- Word Address Register (13 bits) for temporary storage of memory module word address;
- Data Gate (32 bits) and Parity Bits Gate (4 bits) for the routing of information from the data input lines;
- Data Parity Logic for data parity validation;
- Test and Set Logic which provides a memory lock-out feature to be described; and

- Data Register (32 bits) and Parity Bits Register (4 bits) for temporary storage of memory/processor transfers (a local Data Register enables asynchronous MMU/CPU operation).

2. Operation. MMU control lines are listed in table 3. Each of four basic memory operations occupy a dedicated input control line, and a fifth provides a signal path for access request. Four lines provide MMU to processor control communication.

TABLE 3. CONTROL LINE SETTINGS

| Processor to MMU | | MMU to Processor | |
|---|---|---|---|
| Line # | Signal | Line # | Signal |
| 1 | Access Request | 1 | Parity |
| 2 | Read | 2 | Data Ready, or Test & Set not Locked |
| 3 | Write | | |
| 4 | Test & Set | 3 | Test & Set Locked |
| 5 | Change BA | 4 | Address Match |

a. Access request decoding. An End Around Shift Register (EASR), shown in figure 4, sequentially tests the access request lines of all connected processor buses via a series of circular shifts. The EASR contains a shift position for each possible bus connection and a single bit position is set to one (1). An access request gate is associated with each processor bus. Inputs to an access request (AND) gate are the access request line from the processor bus and the value of the EASR position assigned to the bus. An access request is recognized when the set bit of the EASR coincides with a processor bus position for which the access request signal is present. Recognition of an access request effects a temporary halt of the EASR and signals the Switch Control, providing processor bus identification information. The Switch Control connects the memory module to the bus recognized, but access is not yet granted.

The memory module (figure 5) compares the contents of its Bank Address Register (BAR) with the bank address from the address lines (high order five bits). If the compare is equal, access is granted and a bank address match signal is transmitted to the processor. The address match signal is used to reset the access request line, thus other MMUs will not perform a bank address

compare for the recognized processor bus during the remainder of the memory operation. Following the bank address match signal, the module maintains the bus connection to the requesting processor until the memory operation is complete, at which time the EASR is enabled and scanning resumes. The completion of a memory operation is signaled via the data ready control line, or, in the case of an anomaly, via the bank address mismatch line (AMM), test and set locked line (TSL), or the parity error line (P).

Any of the following events constitute a continue scanning command to the EASR.

- Alignment of EASR does not recognize an access request.
- An address mismatch signal (derived from the address match signal described above) is received from the memory module.
- Data ready, parity, or test and set locked signal is transmitted to processor.

In the latter two cases, a disconnect command is issued to the Switch Control.

The depicted operational characteristics of EASR request scanning is the most basic configuration. Implementation of a priority recognition arrangement is feasible, but present criteria do not indicate the necessity.

b. Control decoding. Referring to table 3, control information received by the MMU may initiate operations to read, write, test and set, and change BA. Completion of each operation results in a positive response (in the form of a control signal) from the MMU to the processor. Figure 5 (and figure 6 which is of greater detail) supports the following discussion of the individual operations.

(1) Read. After access is granted to the memory module and an address match signal transmitted to the processor (which resets the access request signal), address parity is checked. Invalid parity results in the transmission of a parity signal to the processor. If parity is valid the word address bits (lower order 13 bits of address lines) are gated to the WAR and used to access one of 8,192 words in the storage array. During the read/restore cycle, the data from memory is validated via parity check and the parity signal to the processor raised if a parity fault is detected. Once validity is ascertained the data (with parity) is transmitted to the processor via the 36 data out lines with concomitant data ready signal. The EASR resumes its scan and disconnection from the memory module occurs after a short delay.

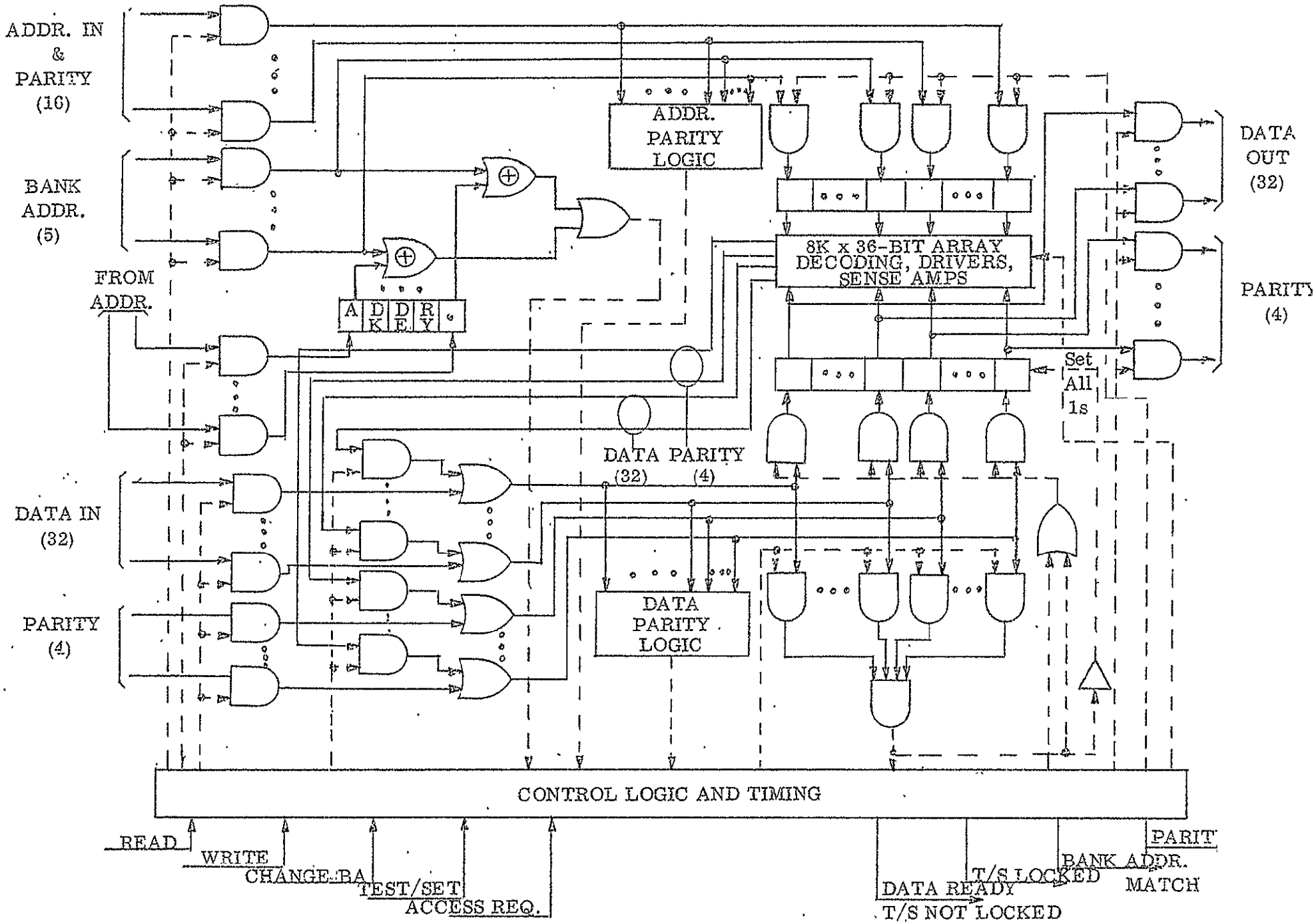



FIGURE 6

MULTIPROCESSOR MAIN MEMORY MODULE DETAIL

(2) Write. The processor/MMU dialog is analogous to the read command with two exceptions. First, following the read half of the clear/write cycle 36 bits are gated from the data in lines to the data register and associated parity bits register. During this transfer the parity is checked; invalid parity will result in transmission of the parity signal to the processor. Second, following completion of the clear/write cycle no data is made available via the data out lines, but the data ready signal is transmitted to the processor to indicate completion.

(3) Test and set. This memory operation provides in one memory cycle for testing a storage variable for zero, setting it to all ones if zero or raising a signal to the processor if not. Thus, complete protection of global data and code may be effected.

After access is granted, the first half of the cycle is equivalent to the first half of the read/restore cycle. The test word which has been fetched from memory will contain all "ones" if "locked." After the test word has been checked for parity errors, the Test and Set Logic detects the presence or absence of all "ones."

- If the test word does not contain all "ones," the Data Register is set to all "ones" and this information written into the test word memory location; the test and set not locked signal (which appears as a data ready signal to the processor) is transmitted.

- If the test word contains all "ones," it is written back into memory via the Data Register and the test and set locked signal is transmitted to the processor.

(4) Change BA. After granting access and checking address parity, selected lines from the thirteen lower order address bits are gated to the Bank Address Register. Changing the BAR resets the bank address match signal. Trailing edge detection logic in Control Logic and Timing in conjunction with the change BA control signal then generates the data ready signal.

## B. Central Processing Units (CPUs)

The major element of the MEC is the central processor. The system structure as depicted in this report is based upon the SUMC and, for this reason, the approach to CPU architectural specification is to summarize a prespecified baseline SUMC definition, and then define necessary and sufficient departures from this baseline. The departures are necessary to enable the

SUMC to function efficiently in a multiprocessor environment; they are sufficient in that, while other features could be added or alternate methods of implementation could be employed, those departures specified herein will support efficient multiprocessor operations.

1. SUMC Baseline. All baseline documents are oriented toward simplex system usage of the SUMC. References 3 and 4 provide brief overviews to the SUMC logic at a functional block diagram level. In addition, reference 3 derives an efficient software-oriented organization based upon an assumed 24 bit word. The organization is depicted through specification of formats for a basic instruction set, register organization, and a stacked interrupt scheme. Since a 32 bit word size is assumed for the MEC CPU and main memory, much of the argument presented in reference 3 is invalid.

References 2 and 5 outline the organization of the SUMC data flow and module functions in addition to specifying the microinstruction word fields and operations. Microinstruction read-only-memory (MROM) sequences for several selected instructions are given in both references to show the microprogramming capability.

Reference 6 gives a rather exhaustive set of instructions proposed for a 32 bit version of the SUMC, while reference 7 offers a conventional CPU-controlled approach to handling I/O (similar to what might be found in several

---

[3] Kennedy, J. R.: Basic Instruction Set for a Proposed 24 Bit General Purpose Spaceborne Digital Computer. Contract Report prepared for MSFC by Computer Sciences Corporation under NASA Contract NAS8-18405, Huntsville, Alabama, August 13, 1969.

[4] Garett, Harrison: Advanced Aerospace Computer Technology. NASA TMX-64504, Research Achievements Review, pp 37-44, Vol. III, No. 11, MSFC, Huntsville, Alabama, 1970.

[5] Eastin, E. I.; Little, G. D.; Romine, M. G.; and Williams, C. A.: MSFC Advanced Aerospace Computer. Contractor Report SP-232-0384 prepared for MSFC by Sperry Rand Corporation under NASA Contract NAS8-20055, Huntsville, Alabama, July 6, 1970.

[6] Thompson, E.; Williams, C. A.; Eastin, E. I.; Little, G. D.: Proposed Instruction Set for SUMC System. Contractor Report SP-232-0405-1 prepared for MSFC by Sperry Rand Corporation under NASA Contract NAS8-20055, Huntsville, Alabama, September 4, 1970.

[7] Williams, C. A.: A Possible Interrupt and I/O Scheme for SUMC. Contractor Report SP-232-0399, prepared for MSFC by Sperry Rand Corporation under NASA Contract NAS8-20055, Huntsville, Alabama, August 14, 1970.

commercial systems). A scheme for interrupt control associated with the I/O capability is also outlined in reference 7.

a. Block diagram and microinstruction format. Figure 7 is a block diagram of the 32 bit simplex SUMC depicted in reference 5. Information is moved, generally from left to right, through the ALU, where two multi-function adders can be used to operate on it, and into the MRU where it can be looped back through the ALU for further operations, stored in SPM, or made available for storage in main memory or output to other external devices. Control of the source of the information, the operations to be performed, and its disposition once it has reached the MRU are all made by the Control Logic and Timing (CLT) under the direction of microinstructions obtained from a fast read-only-memory (MROM).

The baseline format of each MROM word is given in figure 8. Detailed descriptions of fields and subfields are given in references 2 and 5, although they do not agree completely due to the evolutionary nature of the SUMC. Figure 8, excerpted from reference 5, is of a later vintage and therefore is considered as the baseline. A total of 72 bits comprise the full word.

Several areas of interest are worth noting at this point since they will be influenced by departures:

- Only 64 words of SPM are addressable.
- Only "read" and "write" MMU functions are accommodated.
- No registers are provided for efficient program address relocation.
- No registers are provided for MMU access violation detection.
- The capability for condition setting and the associated testing for MROM branch control is weak.
- Field specifiers for direct (CPU/device) input/output control are inadequate.

b. Instruction set and register organization. Several of the previously referenced documents propose various instruction repertoires and register organizations. No specific instruction word formats are claimed to be optimized to a 32 bit word size as a result of analysis methods similar to those followed in reference 3 for a 24 bit word size. For this reason, it is

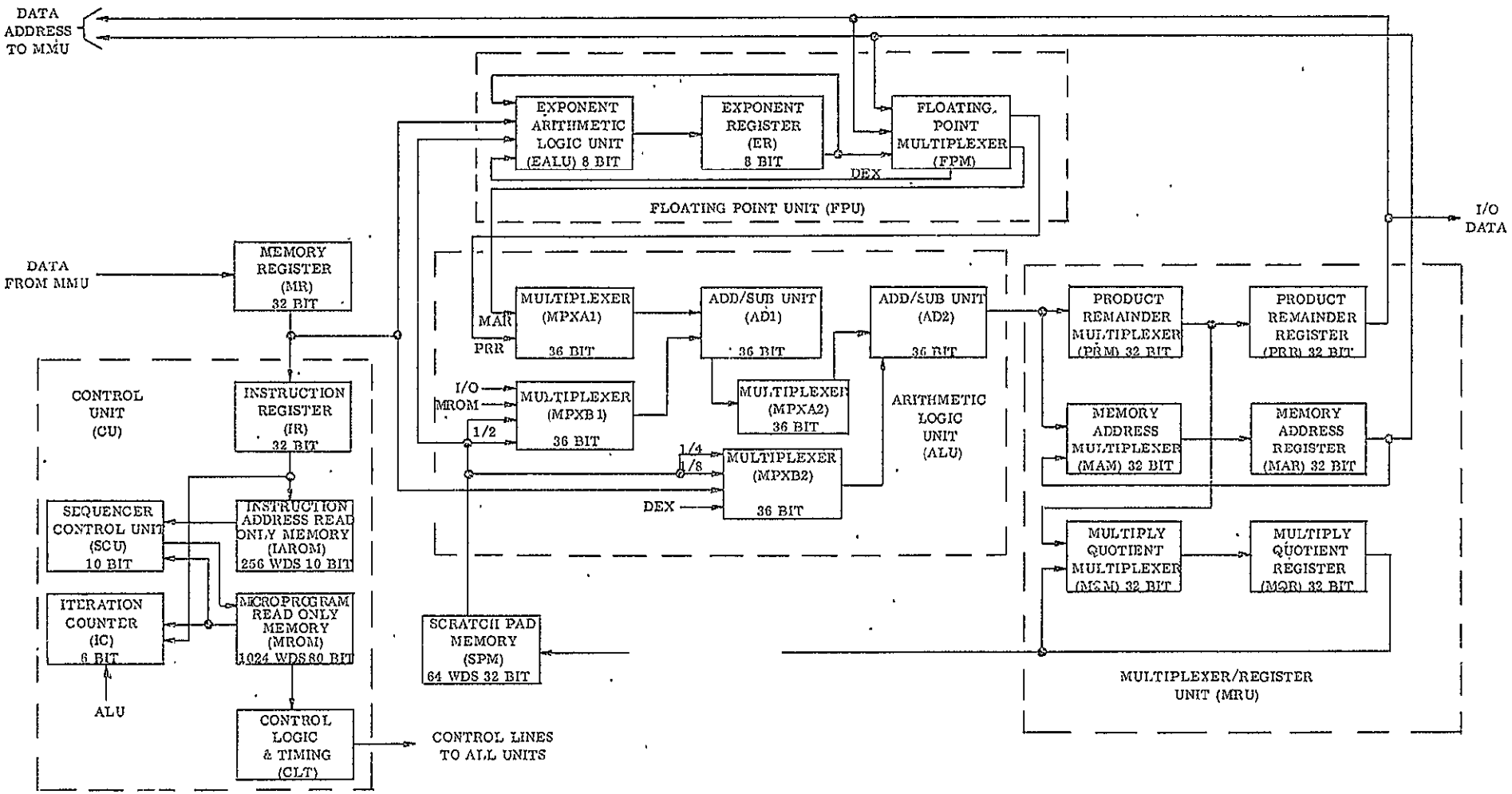



FIGURE 7

SUMC BLOCK DIAGRAM

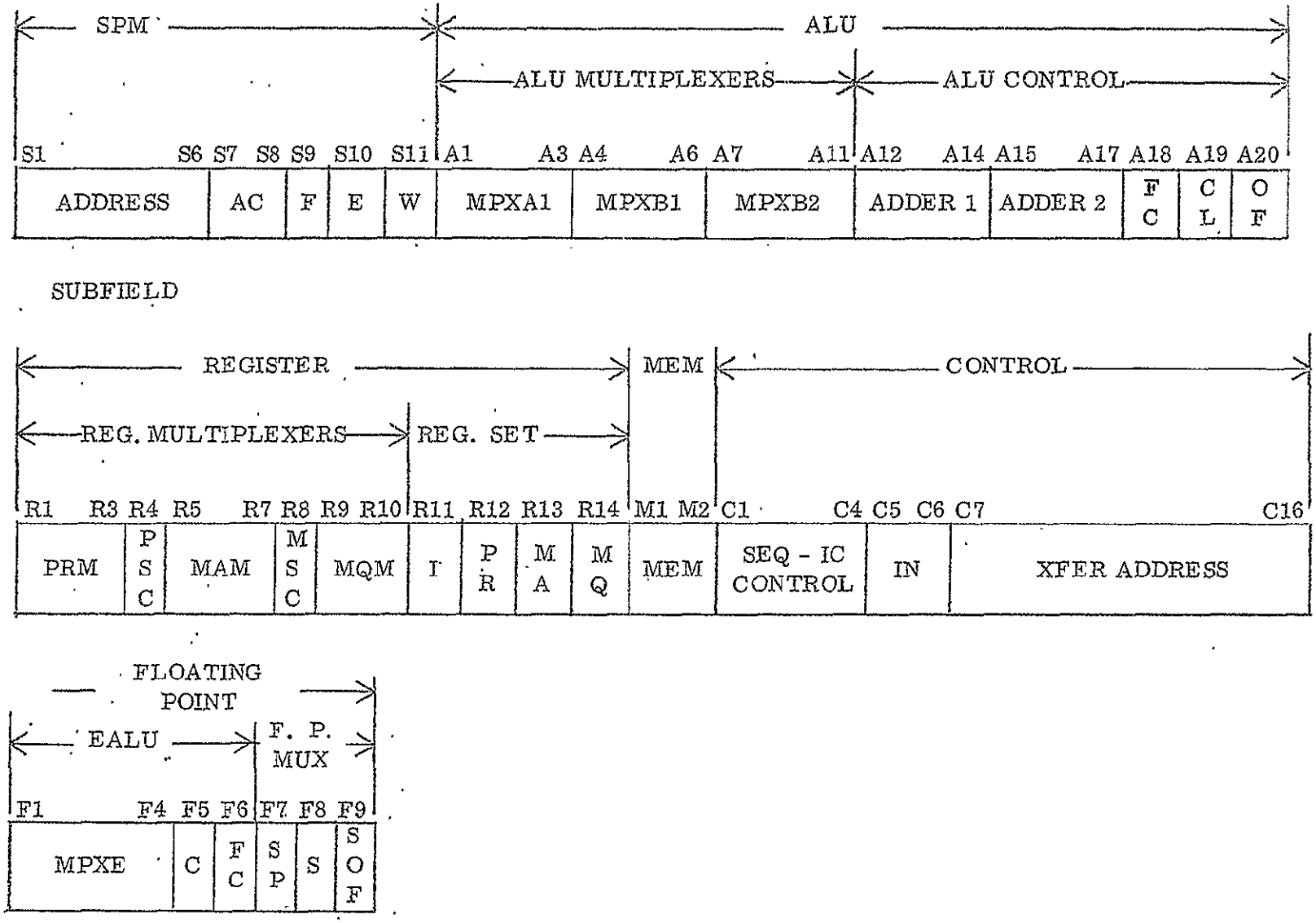



FIGURE 8

MICROINSTRUCTION WORD FORMAT FOR MROM

felt that no optimal baseline instruction set exists. For the purpose of support to subsequent tasks, however, the collective functional capability of all previously specified repertoires is assumed, and the format shown in figure 9 is adopted for memory reference instructions (only). When, for the purpose of estimating program sizes, it becomes necessary to assume a specific repertoire, a specification will be required.

The baseline register set organization is taken to be that of figure 10. This organization was favored by reference 3 and mentioned as a viable candidate by reference 5.

c. Input/output. Baseline candidate definitions of the IOP vary in the accorded capability from that of a conventional direct memory access (DMA) controller to that of simple logic to augment SUMC controlled data transfers. The DMA approach proposed in /7/ adopts the viewpoint that the IOP be designed with minimal capability.

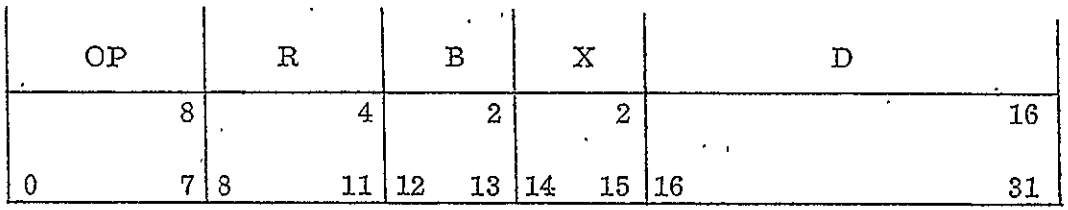
Reference 8 outlines a two-option approach to the control of system input/output. One is referred to as a "Simplex Input/Output Controller" and the other is called a "Combined Free Running and Integrated/Dedicated Controller." Both are defined with respect to simplex system configurations, and both are operationally controlled by CPU issued initiation (called program controlled output [PCO]) instructions, and stored program input/output command sequences (Externally Controlled Output [ECO]) fetched from memory for decoding and execution by the IOP. In addition, both show functional block diagrams illustrating the modem interfaces with a data bus for various channels, and ECO commands for memory/device and device/device transfers and response/transfer monitoring.

The major difference in the two approaches is that, in the first, ECOs and device data/control words are fetched from SUMC main memory whereas, in the second, ECOs and device data/control words are fetched from a "format buffer" (FM) consisting of a 4K local memory. Also, the second option allows for commutated word I/O through the use of a special ECO to address a scratch pad memory whose words are used functionally like a group of index registers.

In the second option, no facility is indicated for writing into FM, therefore leading to the assumption that it is operationally read only. This implies

---

[8] Space Station Newsletter No. IBM/SPE-96. Transmittal of IBM study data from A. J. Kemp, IBM Huntsville, to H. Ness, MDAC-WD, June 21, 1971.




OP: Operation Code
R: Register Address (one of 16)
B: Base Register Address (one of 3; "0" implies no base used)
X: Index Register Address (one of 3; "0" implies no index used)
D: Displacement Address (one of 65,536 MMU virtual locations)

FIGURE 9

MAIN MEMORY ACCESS INSTRUCTION FORMAT

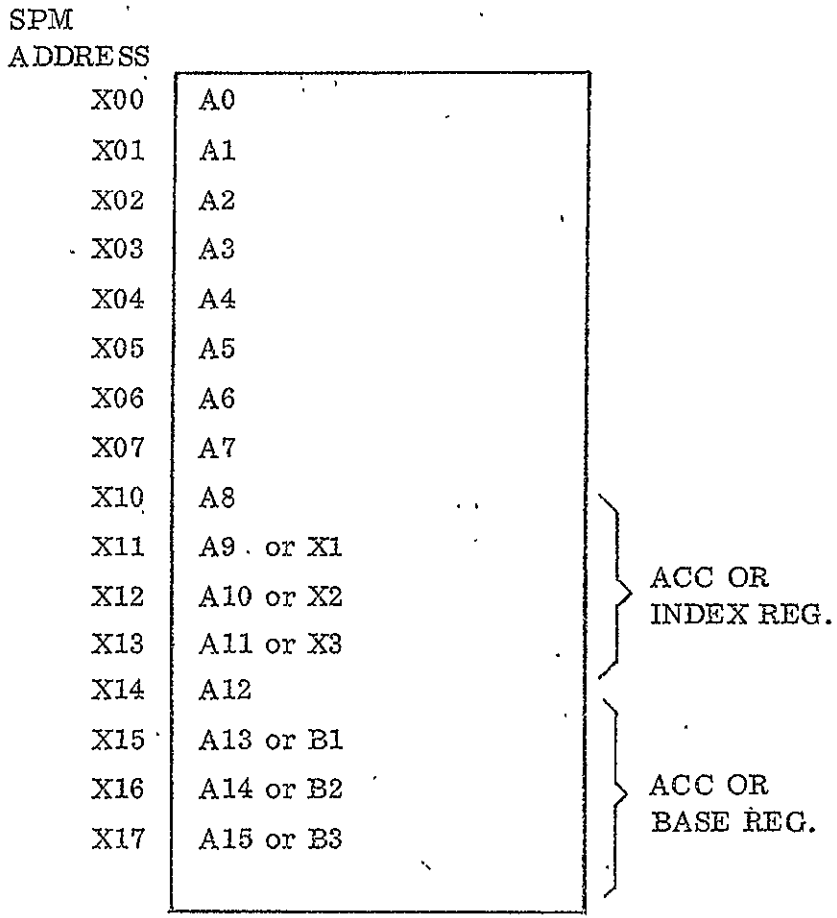



FIGURE 10

PROGRAMMABLE REGISTERS

a prestructured FM content suitable for controlling all data bus transfers. This scheme, although rather inflexible, provides for (or demands, depending on the viewpoint) a preconceived and perfectly organized flow (order, rate and direction) of bus traffic. It appears that a high degree of flexibility could be attained by a provision for program controlled alternation of the source of ECO and device data/control words between main memory and the local FM (the second option does not depict support for main memory-to-bus transfers).

Figures 11 and 12 show the 32 bit bus control instruction candidate formats for the two options. Figures 13 and 14 depict the organization of the two options.

The three candidate I/O schemes outlined above can be summarized as

- Entirely CPU programmed controlled,
- Combined CPU programmed initiation and fetched-from-main memory command controlled (Option 1), and
- Combined CPU programmed and fetched from local (format) memory command controlled (Option 2).

Of these three, Option 2 is felt to offer the strongest baseline from which departures can be made to provide both a desirable degree of flexibility and the necessary functional capability.

2. Baseline Departures.

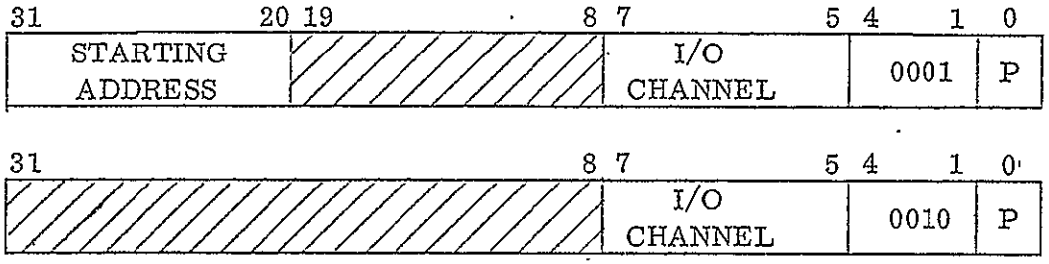
a. Microinstruction fields. Changes to the SUMC microinstruction format are required to support MEC operations. These changes are primarily in the form of expansion to allow for

- Larger SPM
- More MMU functions
- Two modes of CPU operation
- SCU/CPU communications
- Larger MROM

The required changes are briefly outlined as follows:

(1) Add one (1) bit to the "address subfield of the "SPM" field allowing for $2^7 = 128_{10}$ addressable scratch pad memory locations.

PROGRAM CONTROLLED OUTPUT (PCO) INSTRUCTIONS




EXTERNALLY CONTROLLED OUTPUT (ECO) INSTRUCTIONS

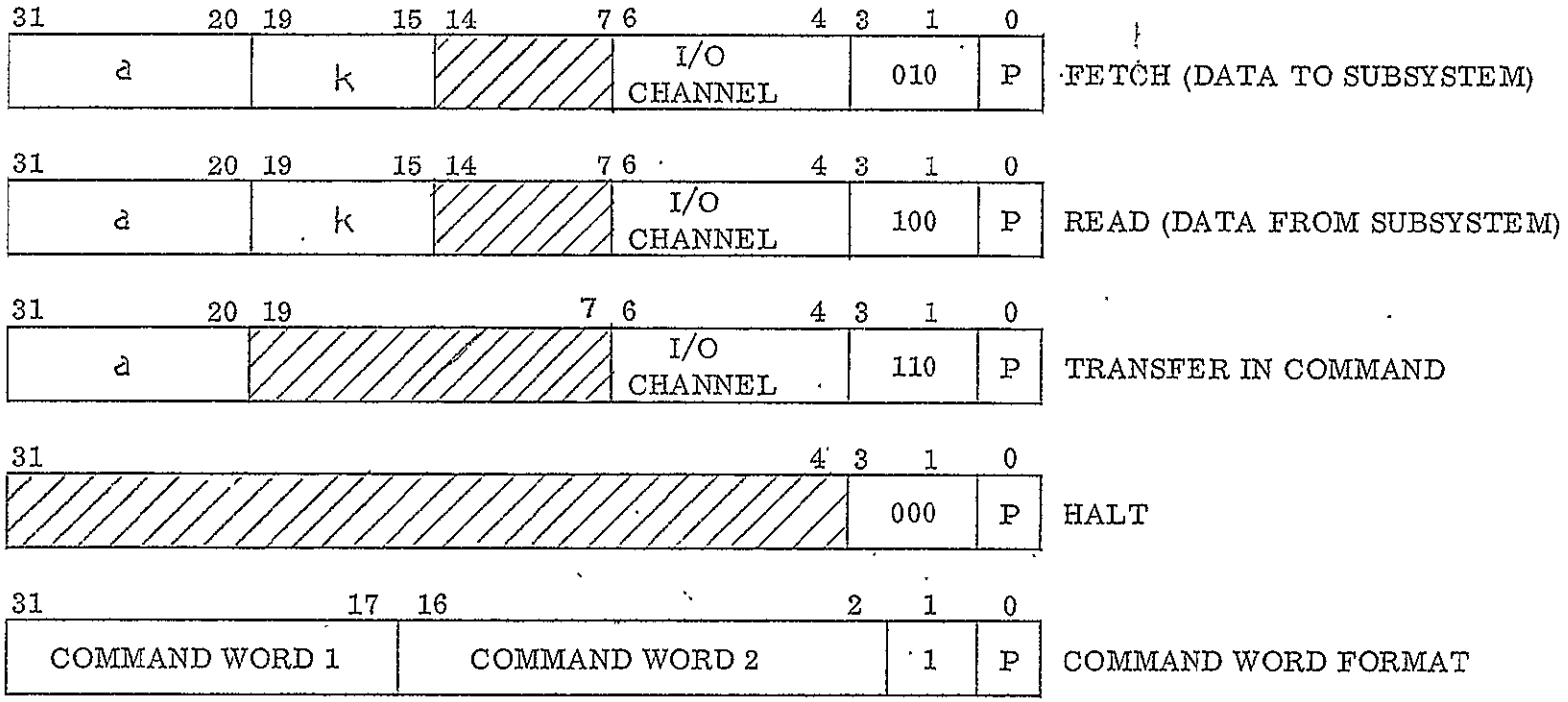
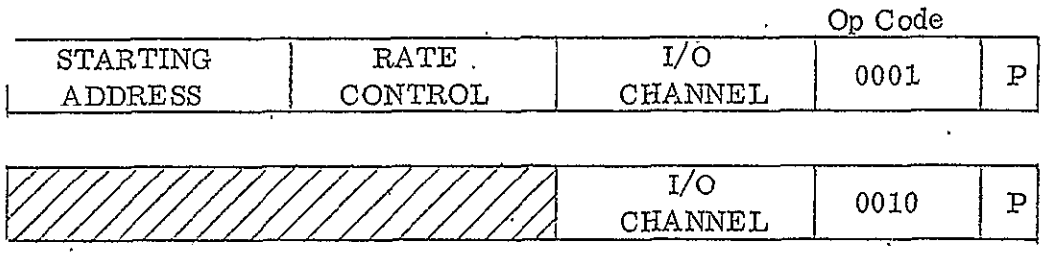



FIGURE 11

OPTION 1 - BUS CONTROL INSTRUCTIONS

PROGRAM CONTROLLED OUTPUT (PCO) INSTRUCTIONS




EXTERNALLY CONTROLLED OUTPUT (ECO) INSTRUCTIONS & FORMATS

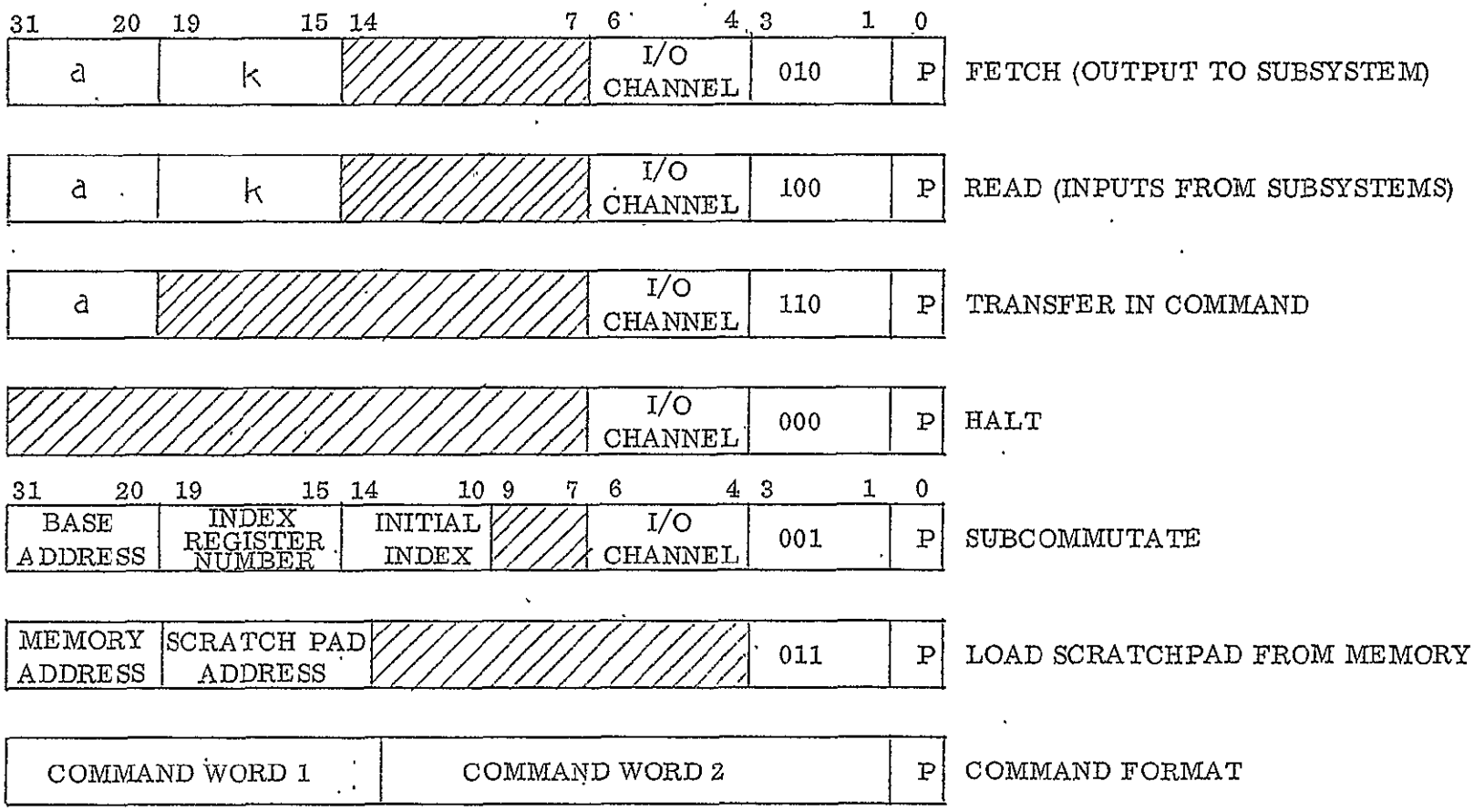
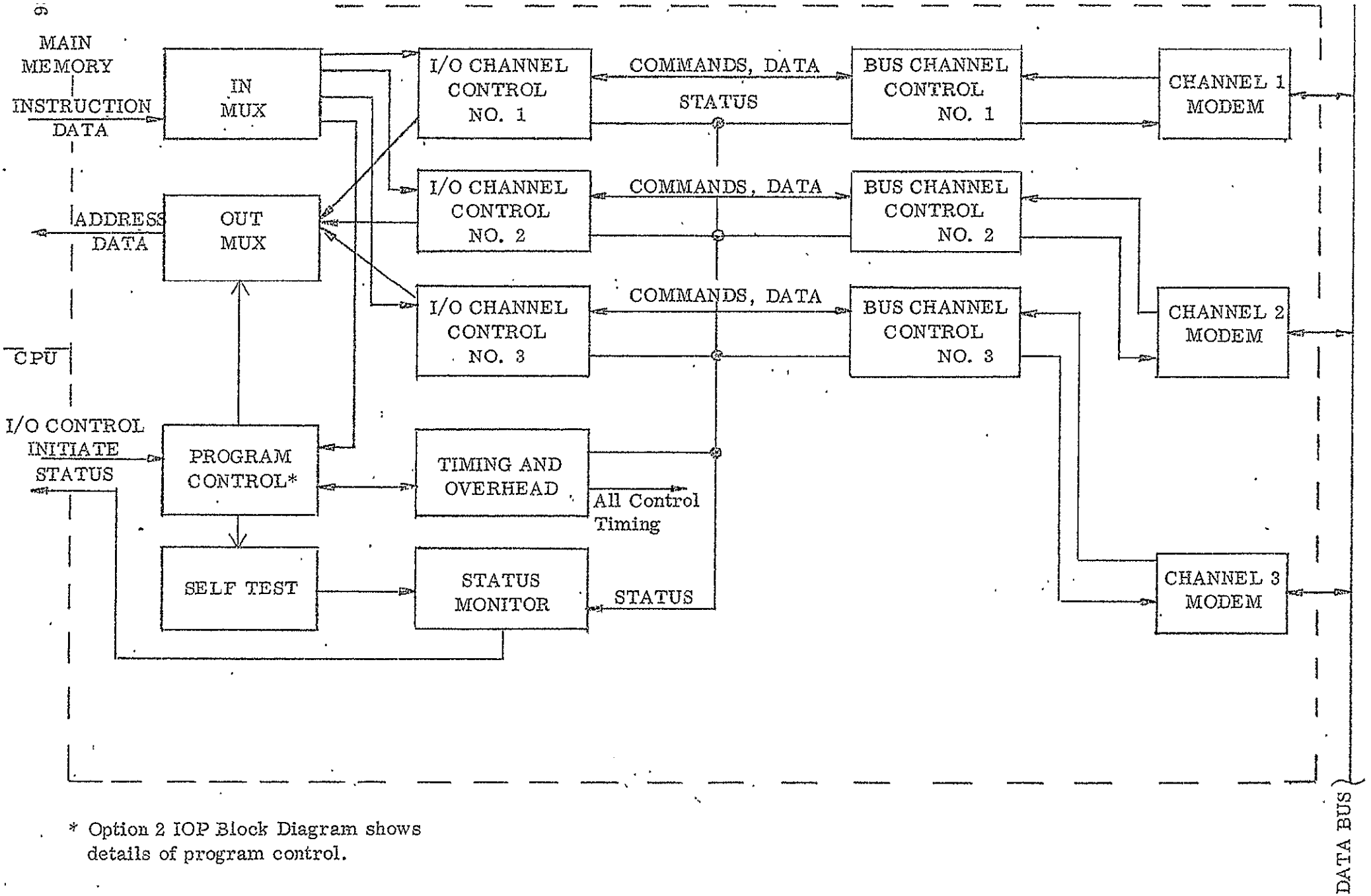



FIGURE 12

OPTION 2 - BUS CONTROL INSTRUCTIONS




* Option 2 IOP Block Diagram shows details of program control.

FIGURE 13

OPTION 1 INPUT/OUTPUT PROCESSOR (IOP)

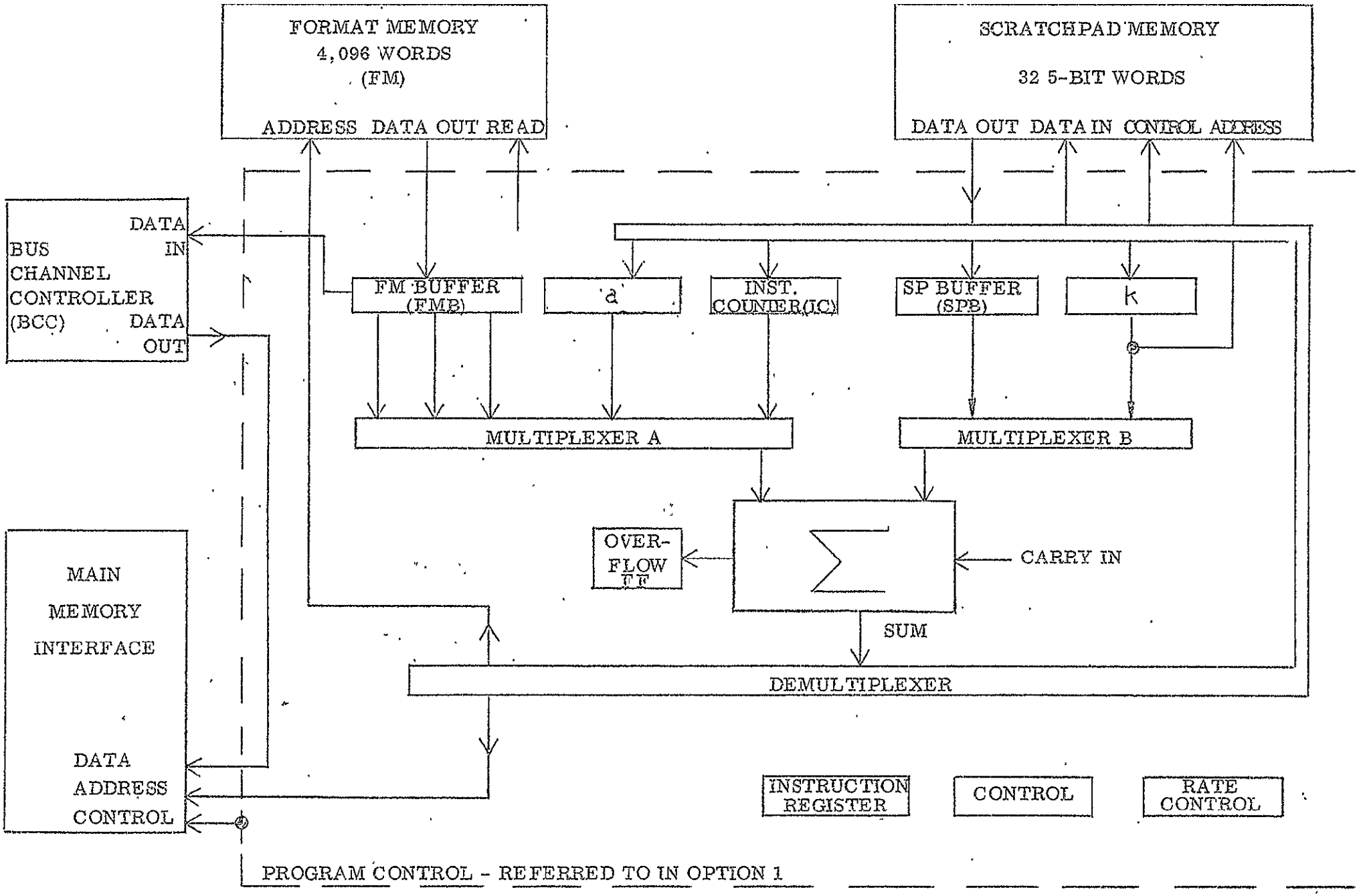



FIGURE 14

OPTION 2 IOP BLOCK DIAGRAM

(2) Define a one (1) bit binary state (flip-flop) register, U, to be located in the CLT module for testing under control of the "SEQ-IC CONTROL" subfield as indicated in (3) and (4) below.

(3) Define a microinstruction bit to be the "MODE" change subfield. A one (1) in this subfield will cause U to be toggled (state changed). A zero (0) has no effect on U.

(4) Add bits (for a total of four [4]) to the "MEM" field to control main memory accesses as follows:

| | |
|---|---|
| 0000 | No access request |
| 0001 | Read |
| 0010 | Write |
| 0100 | Change Bank Address Register |
| 1000 | Test and Set |

(5) Add control and status lines between main memory and CLT as follows:

Status (MMU to CPU/CLT)

| | |
|---|---|
| 0001 | Parity Check |
| 0010 | Data Ready |
| 0100 | Test & Set busy (access lockout) |
| 1000 | Bank Address Match |

Control (CPU/CLT to MMU)

| | |
|---|---|
| 00000 | No access request |
| 00011 | Read |
| 00101 | Write |
| 01001 | Change Bank Address Register |
| 10001 | Test and set |

Note: The main memory access request control line could be eliminated, since "OR"ing the remaining bits provides the required degree of control. However, main memory logic becomes more complex.

(6) Add control line from SCU to CLT to enable detection of an SCU command to CPU.

(7) Add control line from IOPs (one line shared by all IOPs controlled by the CPU) to enable detection of an IOP "poll request."

(8) Add logic to CLT to expand the use of the "SEQ-IC" subfield of the "CONTROL" field as follows:

| | SEQ-IC Subfield | Conditions | Sequencer(S) Action | Iteration Counter (IC) Action |
|---|---|---|---|---|
| (a) | 0000 | (U) = 0 | +1 | None |
| | | (U) = 1 | (M) − S | None |

(b) Add one (1) bit in "SEQ-IC" subfield to support control of branches on the basis of various tests as follows:

Memory Parity Check
Data Ready
Test and Set Busy
SCU Command
IOP Poll Request

(9) For the purpose of software concept verification, testing and validation (CVTV), additional MROM will be required to enable incorporation of debugging capabilities. After CVTV, the additional memory could be removed. Therefore, add one bit to the "XFER ADDRESS" subfield, all IAROM words, and SCU logic to support 2048 MROM words.

b. Main memory access. In a multiprocessing environment where one or more modular memory units are shared, each of the following problems must be addressed:

- Storage allocation for data and CPU processes (programs),
- An addressing scheme which allows each processor to access all available resources,
- Protection of data and processes temporarily local to one processor from all other processors, and
- An access mechanism which provides concurrent utilization, by two or more processors, of one modular unit with minimum delay.

A number of solutions, some of considerable merit, exists for all of the above. Presented here is a paradigm of a system designed to minimize memory complexity, remain compatible with SUMC architecture, and address each problem.

(1) Page addressing. Capability for system expansion frequently dictates that more address lines to memory be established than can be utilized strictly from the portion of the instruction word dedicated to address selection. Earlier studies indicate this to be the case encountered by the SUMC. If 256K words of memory are assumed, 18 address lines are required. It does not seem plausible that 18 bits of each 32 bit memory reference instruction of the SUMC may be dedicated to address selection while maintaining an efficient use of Scratch Pad Memory and providing a large instruction repertoire.

By adding to each address generated by a processor a hardware relocation register, called the Bank Register Low Address (BRLA), which contains the necessary high order bits, this dilemma is resolved. Furthermore, by extending the BRLA to contain additional portions of the address, a solution to the storage allocation problem is approached.

If the BRLA were the same width as the maximum address, each process, once constructed, could be loaded into memory and executed at virtually any beginning location by setting the BRLA to contain the address of that location. Attaining this flexibility may not be commensurate with the cost in terms of SPM storage, memory utilization map updating, and communication required for process dispatching. It is suggested instead that a 13 bit BRLA be utilized, allowing the lower order 5 bits of the address to be generated exclusively by the instruction.

The above arrangement would provide the following organization of a 256K memory distributed among 8K modular units:

- 32 8K banks,
- 256 pages per bank, and
- 32 words per page.

Figures 15 and 16 depict the format of the BRLA and its combination with the instruction generated address, respectively.

Note that storage allocation always begins on word boundaries that are multiples of 32. Conversely stated, at most 31 words between program processes might not be utilized. This possible loss is considered negligible compared to other advantages presented.

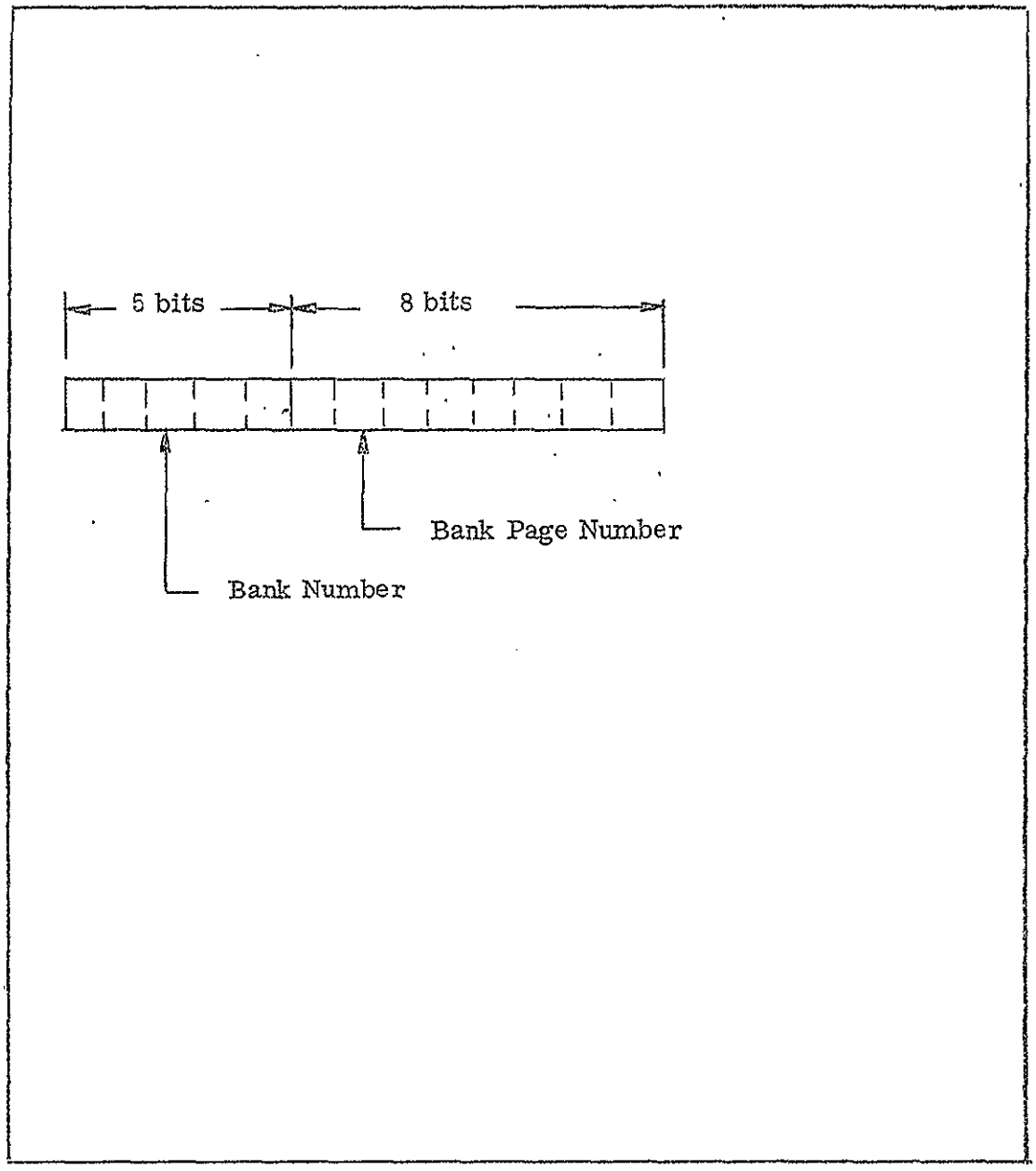



FIGURE 15

BANK REGISTER LOW ADDRESS/BANK REGISTER HIGH ADDRESS FORMATS

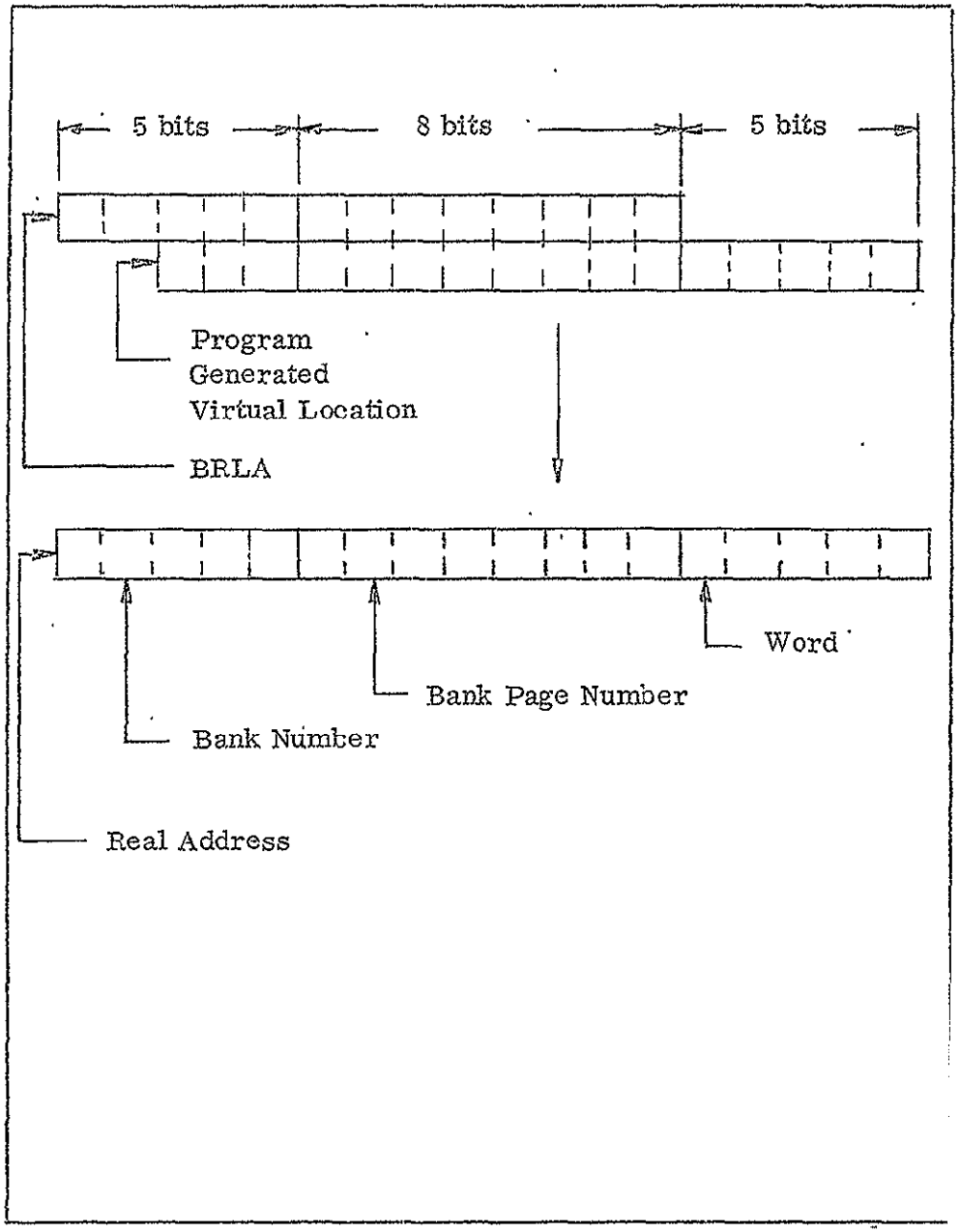



FIGURE 16

ADDRESS GENERATION

Including the BRLA obviates the requirement for lengthy relocation procedures each time a process is constructed (provided internal linkage has previously been accomplished).

The BRLA could be implemented as a location in Scratch Pad Memory for utilization by MROM microinstructions. Additional MROM cycles might be saved on each instruction cycle by implementing it as a hardware register, multiplexed into MPXB2 in the ALU for example. Its addition to the program counter is accomplished only once (during process construction) to be used unchanged until the process is deleted. It must be added to each effective address generated by an instruction.

(2) Memory access violation. Processes that occupy sequential memory locations may generate invalid addresses in only two ways:

- Case 1 - An address less than its lower boundary, or
- Case 2 - An address greater than its upper boundary.

If each process generates addresses relative to zero (the recommended approach) prior to addition of the BRLA, Case 1 may be checked by testing for a negative address immediately preceding addition.

Case 2 implies an additional operation before a check for validity is possible. By including a Bank Register High Address (BRHA) in the organization of Scratch Pad Memory, formatted the same as BRLA (figure 15), it may be subtracted from the final address to obtain a validity check.

The BRHA may, alternatively, be incorporated as a hardware register to minimize instruction cycle time (the recommended approach).

(3) Phased addressing. If more than one processor is executing processes or accessing data juxtapositioned in a single memory unit, the memory unit must alternate memory cycles between processors. An equivalent problem occurs during execution of a re-entrant routine simultaneously by several CPUs. Frequently, memory availability delay has been minimized by providing phased access ports to each memory unit. An alternative can be provided which is simpler to implement and decreases memory access complexity.

If, as in figure 17, the low order two bits (for four bank phasing) of the word number portion of the Memory Address Register (MAR) of the processor are routed to the low order bits of the bank address portion of the memory's address gating register, and all intervening bits shifted lower to compensate, the effect of phasing is obtained. Each set of four sequential

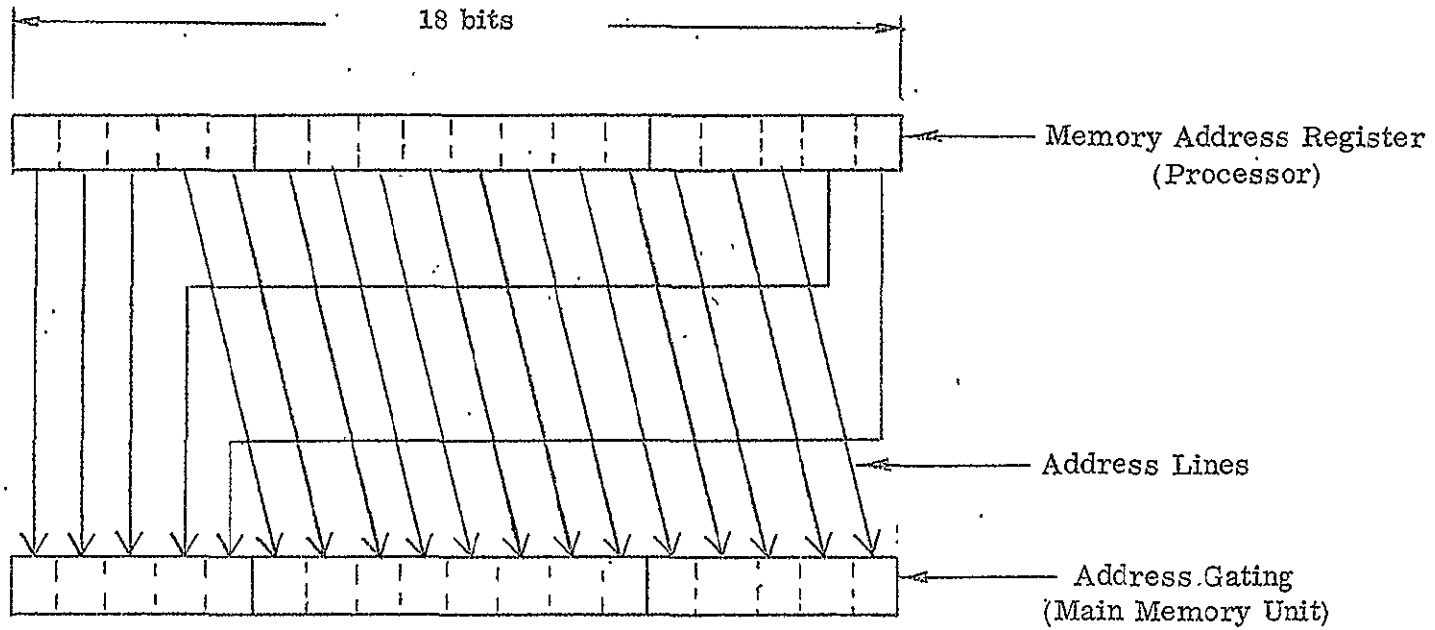



FIGURE 17

MEMORY ADDRESS ROUTING

addresses are distributed among four main memory units rather than contiguously in one unit. Now assume that two CPUs attempt to execute an instruction fetch from the same MMU and visualize the sequence of events. (Figure 18 depicts the storage allocation for "N" processes.) One CPU is granted access to the first MMU and the others must wait. After completing the first instruction fetch the CPU continues to the next MMU, allowing another CPU to access the first MMU. This sequence continues until all CPUs are operating synchronously from different MMUs. Synchronization remains intact until one CPU performs an instruction resulting in non-sequential instruction execution or requires more or fewer memory cycles (data retrieval for example) than the others. At this time an adjustment is made and synchronization is quickly re-established.

Thus, by manipulation (merely cross-connecting) of the address paths, much of the benefits derived from phased access ports may be achieved at no increase in cost or complexity.

It is interesting, however, to examine the benefits which might accrue if an optional non-phased mode were under program control. First, during periods of reduced memory requirements a larger portion of the system could be "shut-down" to reduce power consumption. Second, memory diagnostic procedures for suspected faulty units could be simplified. Third, the element count required for TMR system mode could be reduced if the TMR process were resident in less than four (4) memory units. Finally, a greater degree of system degradation could be obtained with respect to inoperable memory units.

(4) Alternative approaches. The methods derived above were directed at solving memory access problems by shifting the onus of validation to the processor and simplifying the role of memory. A quite reasonable case may be made for relieving the processor of validation checking in order to reduce instruction cycle time and permit a variety of memory structures to be considered independent of the CPU. No attempt is made here to weigh judgment, but it is of interest to assess the costs.

The basic problem is to perform boundary checks of each memory reference by each processor sharing a memory unit. This implies, for each MMU, a set of dynamic boundary registers for each processor and possibly an adder. A fast hard-wired or firmware sequence is required to perform address validation in a non-destruct (or destruct-restore) fashion. MMU/processor controls are required to:

- Set or change selected boundary registers, and
- Signal invalid address.

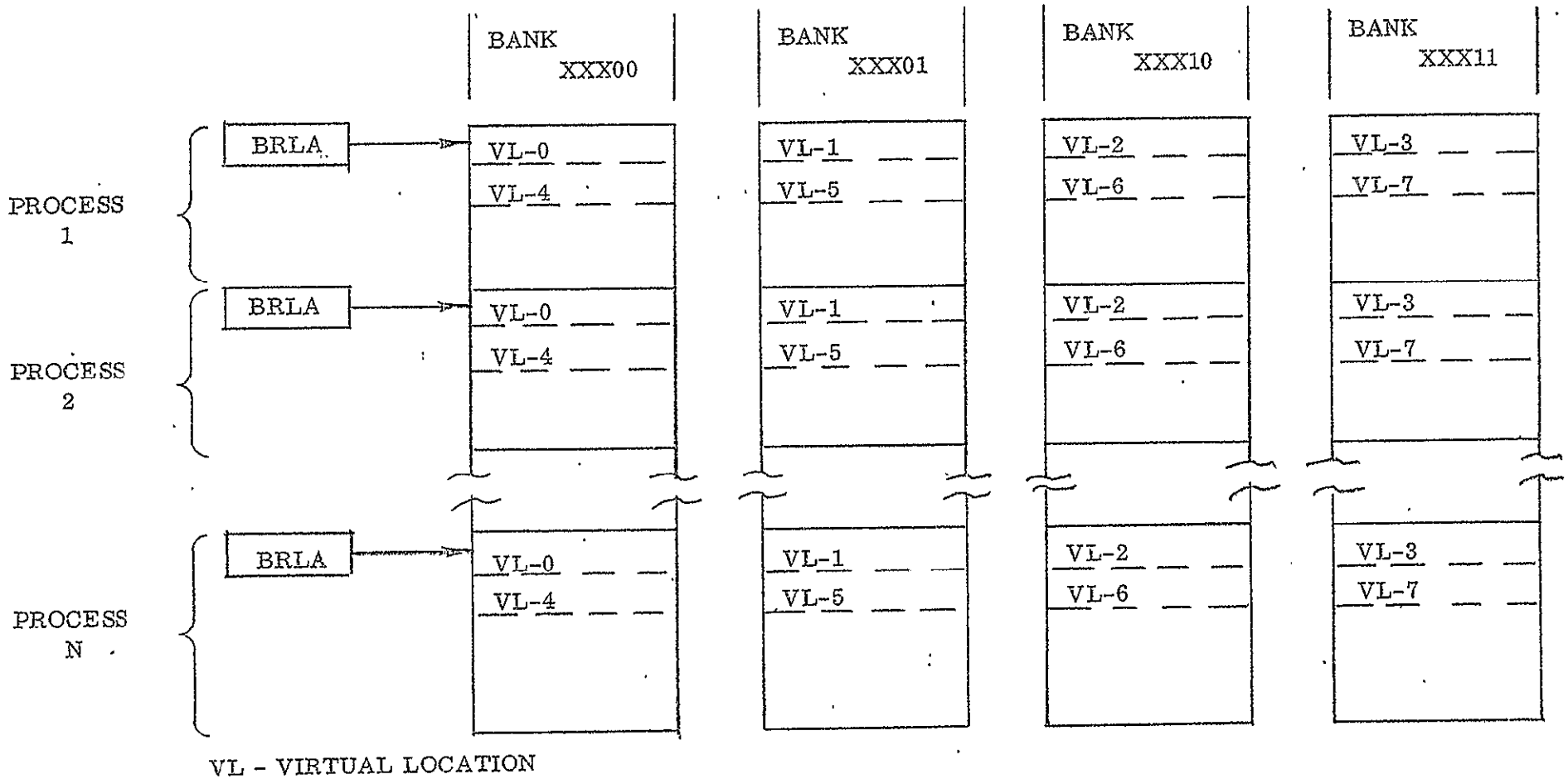



FIGURE 18

PROCESS RELOCATION AND PHASED ADDRESSING

Additional controls that may be of value during process debugging and system diagnostic testing include:

- Disable boundary register, and
- Return (for inspection by a CPU) the boundary register contents.

The benefits accrued at the cost of MMU complexity may be extended beyond reduced instruction cycle time. For instance, memory parity errors may result only after two or more read attempts in order to compensate for transient errors. If the MMU is microcoded to perform the above tasks, an independent self-test diagnostic may be included to assist the system in spares switching decisions and consequent graceful degradation.

(5) Impact on baseline SUMC. The above described approach for memory access could be implemented with microcode alone, thus requiring no changes to the baseline SUMC. However, an increase in operating speed could be obtained by implementing BRHA and BRLA as hardware registers in the SUMC ALU.

d. Process control. The concept of a process and its construction is discussed by Kennedy /9/. Briefly, a process is the sequence of actions performed in order to complete a task. A process may execute code more or less arbitrarily from either executive or application programs and may, in fact, share code with other unrelated processes. Traditionally, the onus of process control and communications between related (cooperative) processes has been entirely the responsibility of the systems programmer. However, the capability provided by a multiprocessor to distribute functional responsibility and the inherent flexibility of microcoded logic can be utilized by the system architect to alleviate the burden as will be subsequently demonstrated. It is necessary to exhibit some basic concepts related to process control.

(1) Process control block. Figure 19 shows a possible structure for a PCB and table 4 explains each entry. Each CPU contains in scratch pad memory (SPM) the PCB of the process for which it is executing code. Processes which have been constructed but are not currently executing are maintained at a central location by an executive routine called the "dispatcher," which is discussed below.

---

[9] Kennedy, J. R.: Executive Routine Primitives and Process Control. Contractor Report prepared under NASA Contract NAS8-18405 by Computer Sciences Corporation, Huntsville, Alabama, March 24, 1971.

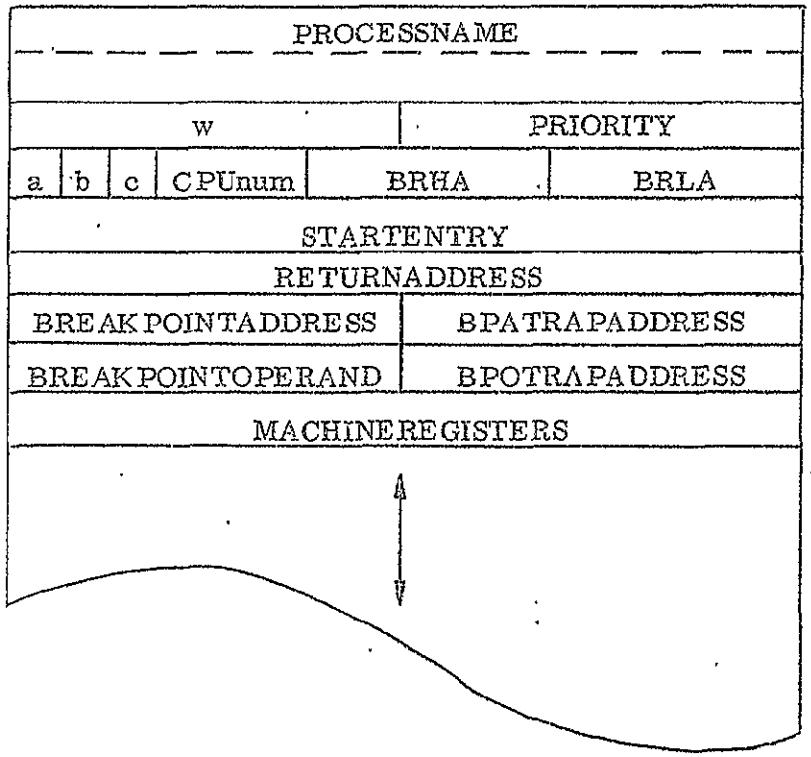



FIGURE 19

PROCESS CONTROL BLOCK

TABLE 4

| PROCESS CONTROL BLOCK ENTRY DESCRIPTIONS | |
|---|---|
| ENTRY | DESCRIPTION |
| PROCESSNAME | Unique name for this process. |
| w | Counter showing number of unserviced START primitives invoked for this process. |
| PRIORITY | Relative process priority. |
| a b c | Three bit process state indicator. |
| CPUnum | Hardware address of the CPU associated, during execution, with this process. |
| BRHA | Bank Register High Address. |
| BRLA | Bank Register Low Address. |
| STARTENTRY | Instruction memory address of first instruction. |
| RETURNADDRESS | Memory address of next instruction in case process activity is stopped; execution will be resumed at this location. Initially has value of STARTENTRY. |
| BREAKPOINTADDRESS | Memory address which, if it becomes the argument of an instruction fetch cycle, will cause an internal processor trap to a predetermined memory address specified by BPOtrapaddress. |
| BREAKPOINTOPERAND | Memory address which, if it becomes the argument of a data fetch cycle, will cause an internal processor trap to a predetermined memory address specified by BPOtrapaddress. |
| MACHINEREGISTERS | A block of words reserved for saving all programmable processor registers when process activity is stopped. Must include all registers depicting process state information. |

(2) Dispatching. Once a process is executing code on a CPU, it may become necessary that the dispatcher seize the CPU for assignment to another, higher priority process. The act of seizing the CPU is called a "preempt" dispatcher action. Any mechanism that effects this task must preserve the current state of the program counter and volatile machine registers. Space in the PCB is reserved for this contingency. Additionally, the dispatcher must retrieve the PCB of the halted process and allow it to compete for CPU time. The act of assigning a process to a CPU is called a "dispatch" action. Clearly the mechanism for "dispatch" is the inverse function of "preempt."

(3) Process states. A process executing code on a CPU is said to be in the "running" state. A process not executing code but competing with other processes for CPU time is in the "ready" state. A process that has been constructed but is not competing for system resources is in the "idle" state.

After a process enters the "running" state, internal conditions may dictate that it not proceed until the occurrence of a specific external event. It may then request that its state be altered until notified by a cooperative process to continue. This interim condition is referred to as the "waiting" state.

A process in any of the above states may be suspended by a cooperative process for examination, alteration, or debugging. For this reason, each state has a companion "suspended" state. A process remains suspended until released by the cooperative process. Table 5 enumerates the salient points concerning process states.

(4) Process state transition. A process may proceed from one state to another by either of two events:

- Dispatcher action ("preempt," "dispatch"), or
- Execution of certain primitive functions (implemented as SUMC instructions) by the affected process or a cooperative process.

Figure 20 illustrates the relationship of the dispatcher and primitives to state transition. The START primitive increments the "w" variable in the PCB which implies a direct transition from the "idle" state to "ready," or subsequent intervention when the process would normally proceed from "running" to "idle." The "w" variable may also serve as a barometer of the workload backlog as detailed in the above cited report /9/.

TABLE 5

| PROCESS STATE DEFINITIONS | |
|---|---|
| STATE | DEFINITION |
| Idle | Process has been constructed but is not currently competing for system resources. |
| Ready | Process is competing for system resources but is not currently executing on a CPU. |
| Running | Process is executing instructions. |
| Waiting | Process has discontinued execution while awaiting an external event. |
| Suspended | For each above state there exists a companion suspended state to or from which a process may revert subject to the action of a cooperative process. |

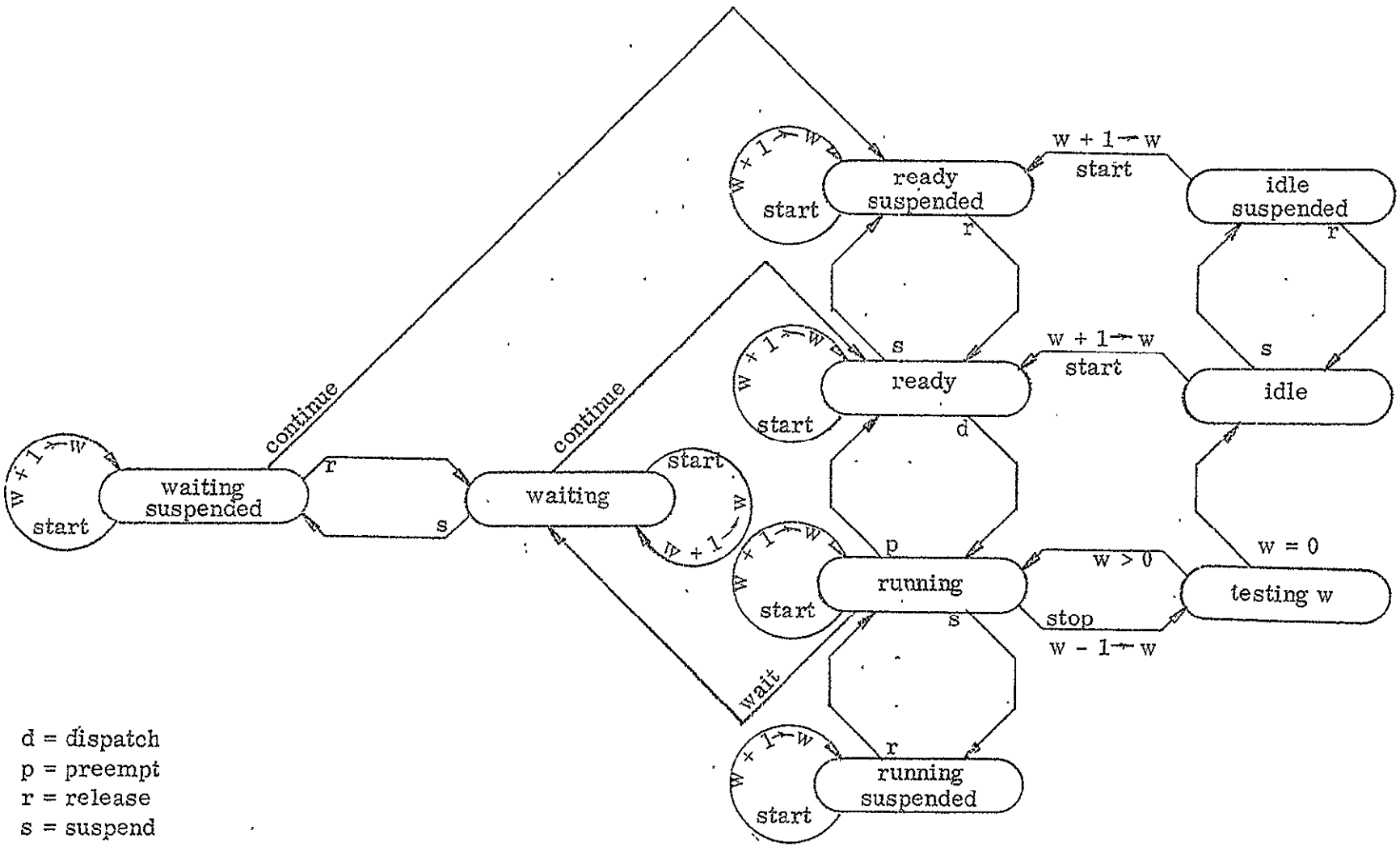



FIGURE 20

PROCESS CONTROL STATE DIAGRAM

A STOP primitive executed by a process in the "running" state decrements the "w" variable. If w becomes zero the process proceeds to the "idle" state; if not it returns to the "running" state.

A process, cognizant of a requirement for some external action (such as I/O), may request transfer to the "waiting" state by executing a WAIT primitive. The "waiting" state is terminated by the performance of a CONTINUE primitive by a cooperative process.

SUSPEND and RELEASE primitives may be executed only by cooperative processes and effect state transitions between companion suspended, non-suspended states described above.

Process termination is effected via ABORT or EXIT primitives. EXIT may be used only for process self-termination. ABORT is available for either self-termination or external termination by a cooperative executive process cognizant of an anomaly. Either connotes transition to a temporary "terminate" state prior to subsequent process deletion. In case of ABORT, additional failure analysis procedures are implemented. For the purpose of simplicity, ABORT and EXIT primitive action is omitted from figure 20.

An additional comment is in order with reference to figure 20. The box labeled "testing w" is not a process state but an intermediate step in the transition from "running" to "idle."

(5) Implementation. Each primitive discussed can be accomplished by manipulation of a process PCB and the transfer of the PCB from the CPU to the system control unit (to be discussed) or vice versa. Thus, at the cost of some microcode logic and shared functional responsibility, a significant attenuation of system overhead can be achieved.

Each primitive is associated with a unique CPU to SCU command (or request) that is transmitted upon execution and is followed by pertinent data. A minor variation of this procedure is invoked by the STOP primitive. The "w" variable is decremented by the CPU and tested for zero, with a command to the SCU resulting only if the value is zero. A detailed discussion of SCU response is given in the section on the system control unit.

Relatively few unique SCU to CPU commands are required for the SCU to perform dispatcher actions and assist during primitive execution. CPU responses to SCU commands are as follows:

(a) Preempt command. In addition to supporting the dispatcher "preempt" action, the preempt command is transmitted to a

CPU (under certain conditions) during execution of a SUSPEND primitive. If the object process is in the "running" state the CPU response is:

- Delay until system is in the user mode,
- Do not fetch next user instruction,
- Complete all pending I/O (where complete may imply abort or other action),
- Save PC in returnaddress field of PCB,
- Send PCB to SCU, and
- Stop with CPU in user mode (where stop implies a micro-instruction idle loop, awaiting the next SCU command).

(b) Dispatch command. This command assists in execution of the RELEASE primitive if the object process is in the "running suspended" state in addition to supporting the execution of the dispatcher "dispatch" action. The CPU response is:

- Receive PCB from SCU,
- Load PC from the returnaddress field of the PCB,
- Load BRLA and BRHA from the PCB, and
- Execute the instruction fetch routine.

(c) Increment w command. Execution of a START primitive for an object process requires that the "w" variable be incremented. If the object process is in the "running" state, the SCU must signal the CPU to effect this change. The CPU response is:

- Discontinue fetch next instruction routine,
- Add 1 to w field of PCB, and
- Continue fetch next instruction routine.

e. Input/output. With regard to CPU functions in support of system I/O, the selected baseline provides for two program controlled output (PCO) instructions as shown in figure 12. Also, simplex system operation only (single IOP) was considered. Therefore, additions to the baseline related to CPU functions take two forms: CPU functions required to communicate with multiple IOPs; and a broader PCO instruction specification to allow control of more IOP functions.

(1) SUMC to IOP communication. Control of the IOPs is effected by transmission of control signals and information over the IOP-CPU control buses (II and IO) noted in figure 2, Uniform Full Non-Dedicated Structure. Data are then transferred to the peripheral devices via the data buses.

Generation of a data transfer sequence is initiated by the recognition by the CPU process of an I/O command known as a Program Controlled Operation (PCO). This PCO must be translated into a format intelligible to the IOP and transferred to the IOP via the II for execution utilizing External Control Output (ECO) instructions. In the transfer of data the CPU must resolve conflicts that may arise as the subsystems compete for CPU cycles. To resolve the competing demands within the baseline SUMC capabilities, a poll-response interaction of the CPUs and IOPs has been recommended.

For the CPU to engage an IOP in a control dialog the following sequence of operations must occur:

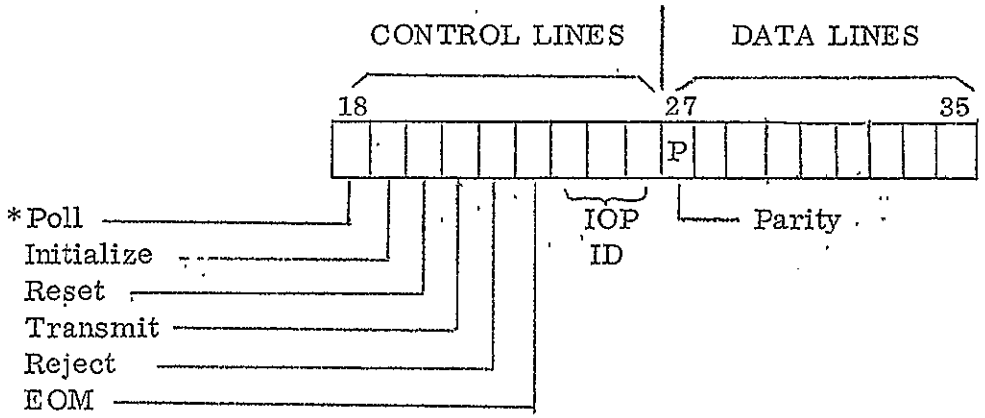
- A CPU raises the POLL line to the Control Logic and Timing section of each IOP. This signals each IOP to expect an address to be transmitted. Recognition is effected before the next FETCH.

- The CPU then transmits the denoted address to all IOPs.

- Each IOP examines the address, comparing it with its own designated address. If the addresses generate a mismatch, the IOP returns to the MISMATCH state. If the addresses match, the IOP transmits ACK and prepares to receive control information. The control sequence can then be sent by the CPU.

Parameters transferred between a CPU and an IOP are shown in figure 21, illustrating parameters required in the handshaking sequences utilized in control of the IOP by the SUMC (CPU). These parameters are defined in table 6.

OUTPUT FROM CPU PRODUCT REMAINDER REGISTER VIA II




*Poll signal is routed to all IOP CLTs.

IOP OUTPUT FROM PRR VIA IO

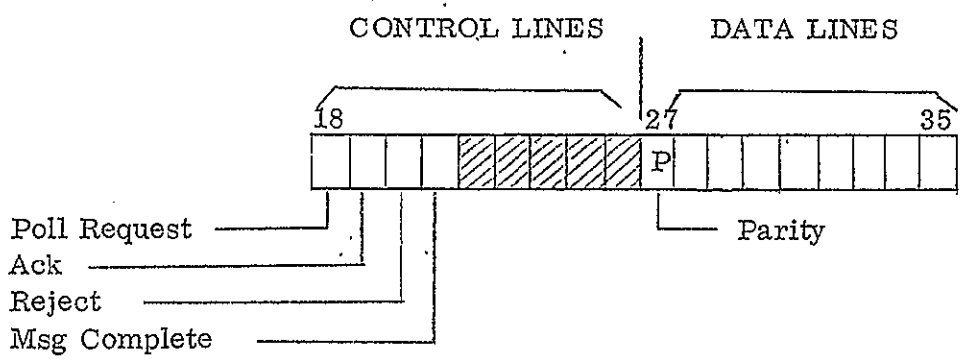



FIGURE 21

CPU CONTROL BUS COMMUNICATION OUTPUT PARAMETERS

TABLE 6. CPU CONTROL OUTPUT PARAMETER DESCRIPTION

| CPU/IOP Output Parameter | Description |
|---|---|
| CPU | IOP Response |
| Poll | Prepare to receive CPU control commands |
| Initialize | Enter Ready state* |
| Reset | Enter Idle state* |
| Transmit | Send one 32 bit word to CPU |
| Reject | Error |
| EOM | End of Message, Mismatch IOPs reset CPU Busy marker |
| IOP Address | Each IOP compares this address with its own and enters either Match or Mismatch state. If Mismatch must set CPU Busy marker. |
| * IOP State Diagram, figure 43 | |

(2) PCO instruction specification. Baseline departures in this case can be thought of as an enhancement of I/O capabilities in the following areas:

- Communication from Main Memory to the data bus,
- Providing the capability to write in the local store (Format Memory) of the IOP in order to revise ECO storage and allow adaptive control of I/O sequences,
- Implementing the capability to retrieve ECOs from Main Memory for execution by the IOP, and
- Expanding the set of PCOs to permit more diverse directions to the IOP from the CPU. Additionally, the addressing capability of the PCOs denoted in /8/ allows an address range of 0-4095 words. While usage of a base register in the address calculation will expand this capability, dedication of unused subfields in the PCO words permit standard SUMC base and index modification address computation.

Considering the preceding factors, the baseline START and STOP PCO instructions are augmented to provide the following I/O commands (defined further under Special Instructions):

- START (BEGIN), initiate an I/O Sequence;
- TERMINATE (STOP), terminate an I/O Sequence;
- GET STATUS, transmit status to CPU;
- INPUT DIRECT, transmit one computer word to the CPU;
- OUTPUT DIRECT, transmit one computer word to a peripheral device; and
- DIAGNOSE, initiate diagnostic process.

The preceding PCO instructions provide the SUMC with the capability to perform the following categories of operations:

- Initiate/Terminate I/O operation of a peripheral device.
- Input the contents of a selected group of status indicators.
- Perform single word transfers between a SUMC scratch memory location and a designated peripheral device.
- Initiate peripheral and IOP diagnostic procedures.

f. Configuration control. By way of summary, the operational aspects of configuration control, as applied to the MEC of figure 1, are outlined here. The capability of the basic scheme is unchanged from that outlined in /1/. However, a significant mechanization change is incurred through a division of responsibility between the SCU and CPU elements.

Configuration control consists of CPU-executed program control that constructs or selects a system map referred to as a setup map (SM). After construction, the SM is transferred over the SI/SO buses from the CPU to the SCU. The SCU then suspends execution of all CPUs, and uses the SM to direct the setting of the various switches connecting element plug positions to buses. The contents of the SM is retained by the SCU and, in this retained form, is known as the action map (AM).

The AM serves as an updatable indicator of not only the structure of the system but also the unique identification of all elements comprising the structure. The actions of switching-out failed elements and switching-in replacement spares is used to update the AM and retain associated element status indications.

Once all SM-indicated switching actions have been accomplished, the SCU commands all CPUs represented as active, connected CPUs to fetch their next instruction from a prespecified location in the MMUs, thus transferring control to the CPU executive(s) for process initiation under the newly established system structure.

Most of the configuration control related actions of the CPU are concerned with building a SM and therefore do not imply communications between the CPU and SCU. Some actions do, however, require SCU cooperation. CPU initiated communications with the SCU are known as "requests" and are as follows:

(1) Switch and jump request. This request is made as a part of CPU execution of the SWJ instruction.

(2) Disconnect element request. This request is made as part of CPU execution of one of the following instructions: SOC, SOM, SOIB, SOII, SQV.

(3) Copy connect request. This request is made as part of CPU execution of the following instructions: CMM, CCC, and CII.

(4) Configuration status request. This request is made as part of CPU execution of the following instructions: SCC, SCP, SCG, SMC, SMP, SMG, SIC, SIP, SIG, and SBG.

The response of the SCU to each of these requests and a description of configuration control related instructions are covered elsewhere in this report.

SCU initiated communications with a CPU are known as "commands" and are described, by way of the response of a CPU, as follows:

(5) Executive transfer command. A receiving CPU responds to this command by taking these actions:

- Do not FETCH next instruction.
- Receive MMU transfer address.
- Place it in executive mode program counter (PC) word in SPM.
- Set CPU mode to executive mode.
- Execute FETCH microroutine.

(6) Receive VDSC error indicators. When a VDSC indicates an error condition to the SCU while operating in a redundant system configuration, the SCU sends this command to the redundant CPUs. The CPUs respond with these actions:

- Do not FETCH the next instruction.
- Receive the status word from the SCU.
- Simulate a "redundant operation failure" interrupt to notify the executive of a failure.

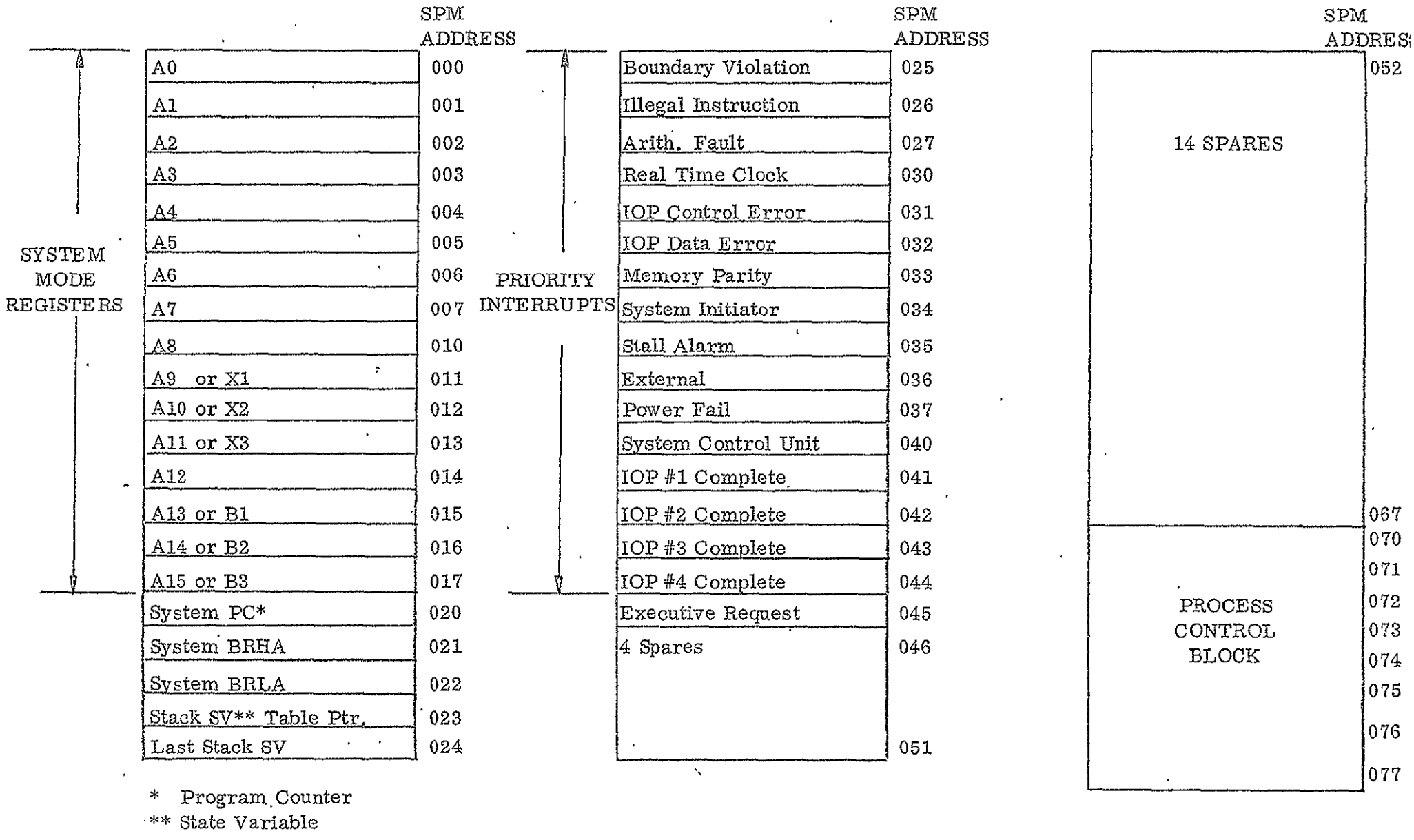
g. Scratch Pad Memory organization. Previous experience indicates that Scratch Pad Memory (SPM) utilization is less than optimum if its organization is consigned in part to software. If the software is permitted access to SPM arbitrarily, and at some point after implementation a reorganization of SPM is effected, much reprogramming will be necessary. For these reasons, SPM should be addressable only from dedicated instruction fields (such as register designation) or implicitly via special instructions. It is requisite under an SPM organization directed by this philosophy that each cell be associated with a specific function. Cell assignments are based on frequency of use and occasionally on response requirements.

A minimum of $128_{10}$ locations are required for allocation of variables that occur in the above two classes. The following discussion is based on an SPM of this size. Specification is partially complete and space is available for expansion.

(1) Major sections. Figures 22, 23 and 24 depict a candidate SPM organization. Locations $0\text{-}63_{10}$ are "system mode dedicated," that is, addressable only when the CPU is in the system mode. Cell assignments in this area represent functions unavailable to instructions executed in the user mode.

Locations $64_{10}\text{-}82_{10}$ are "user mode related." Although accessible during system mode operation, they provide the register set and variables primarily referenced during user mode operation. A similarly ordered set of SPM locations is defined at the beginning of the system mode dedicated section.

Locations $83_{10}$ to $127_{10}$ form the "non-dedicated" section. Assignments to this area represent functions or events that can occur (irregardless of CPU mode) in a more or less stochastic fashion. Examples of this are an IOP status message or the occurrence of a CPU "jump" instruction.




* Program Counter
** State Variable

FIGURE 22

SYSTEM DEDICATED SCRATCH PAD MEMORY

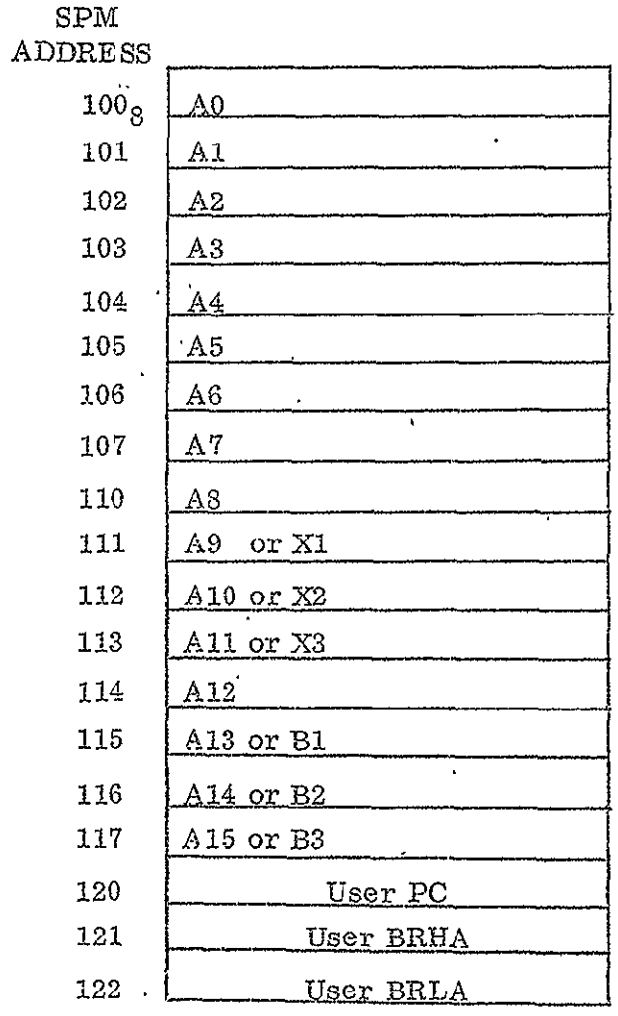
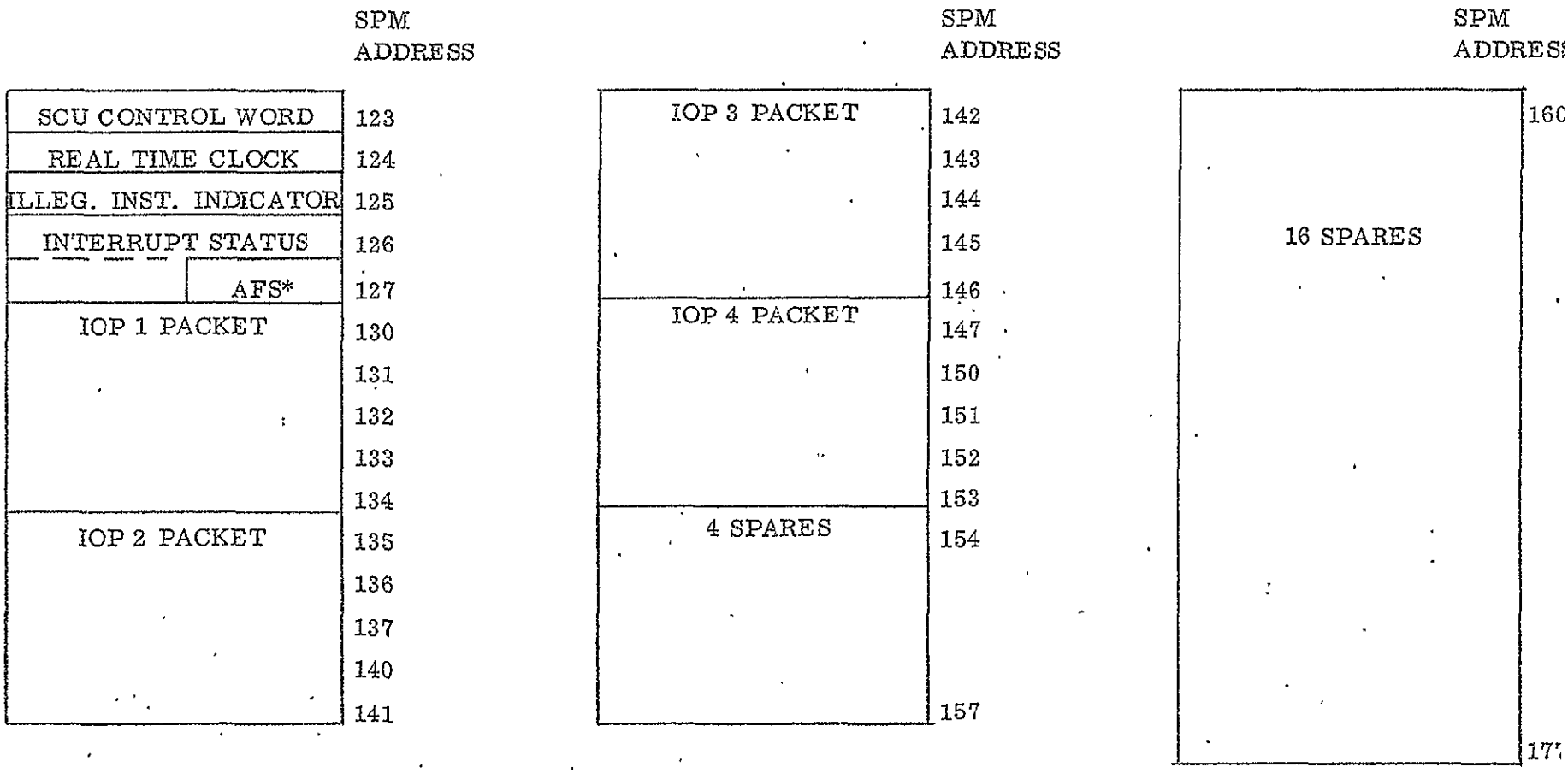



FIGURE 23

USER RELATED SPM SECTION




* ARITHMETIC FAULT MASK AND STATUS INDICATOR

FIGURE 24

NON-DEDICATED SCRATCH PAD MEMORY

(2) Register organization. The format definition for memory reference instructions presented earlier in this section (figure 9) precludes a detailed discussion of the number and types of registers. Overlap between accumulators and base registers and between accumulators and index registers was judged essential since indexing algorithms are frequently derived using arithmetic procedures. Orientation of the register sets relative to the beginning of SPM and to each other was dictated in part by the "OR"ing procedure used to combine the SPM address field of the microword instruction with the offset obtained from the register designator field of the CPU instruction.

A separate (and congruent) register set is allocated for system mode operation. This provision reduces the (software) overhead entailed by mode changes. Evidence does not indicate that the optimum system register set is congruent to the optimum user register set, but a trade study determining the optimum system register set is beyond the scope of this report. A non-congruent system register set requires the reorganization of the format of a large number of CPU instructions, resulting in a vastly different instruction set for system mode use which is costly in terms of microcode requirements. Nevertheless, if further research indicates unusual benefits, a system register set may be defined at a later date.

(3) Interrupts. The sixteen (16) interrupt levels depicted in figure 22 are arranged in order of assumed priority. Definition of software alterable levels is possible, but there is presently insufficient justification. A rearrangement of the priorities is feasible for each mission, partially negating the benefits of a more flexible priority interrupt structure.

Priority within the I/O interrupt levels is based on the type of I/O involved. I/O directly between the device and the CPU is given highest priority since it can be expected to be of low volume and is the most likely form of astronaut/pilot command communications during manned missions. Device to device I/O is given lowest priority with the assumption that it is the least likely to precipitate process idle time prior to completion. Within each I/O type, input has uniformly higher priority than output.

Other SPM locations associated with the interrupt structure are as follows:

- A five (5) word communications packet for each IOP,
- A fifty (50) bit interrupt status indicator,
- A fourteen (14) bit arithmetic fault mask/status indicator, and
- An SCU control word.

An IOP packet (figure 25) is used to communicate to the CPU the current IOP status, associated channel status, the last channel operation executed, requests for service, and pre-selected data items. The "P" bit of the packet (first word, bit 0) is always received as a one (1), and reset by the microcode interrupt service routine when service (by the system Exec) is granted or such service is deemed unnecessary (i.e., "P" is a "protect" bit that prevents the packet from being destroyed before its contents are accessed).

The formats of the interrupt status and arithmetic fault mask/status indicators are illustrated in figures 26 and 27. A note of explanation is in order concerning the "queued" status entries. An interrupt is queued for subsequent service after a request is received that cannot be immediately processed due to its priority level or "disarmed" status. A complete definition of "enabled," "disabled," "armed," and "disarmed" is delayed until the definition of instructions associated with interrupt processing.

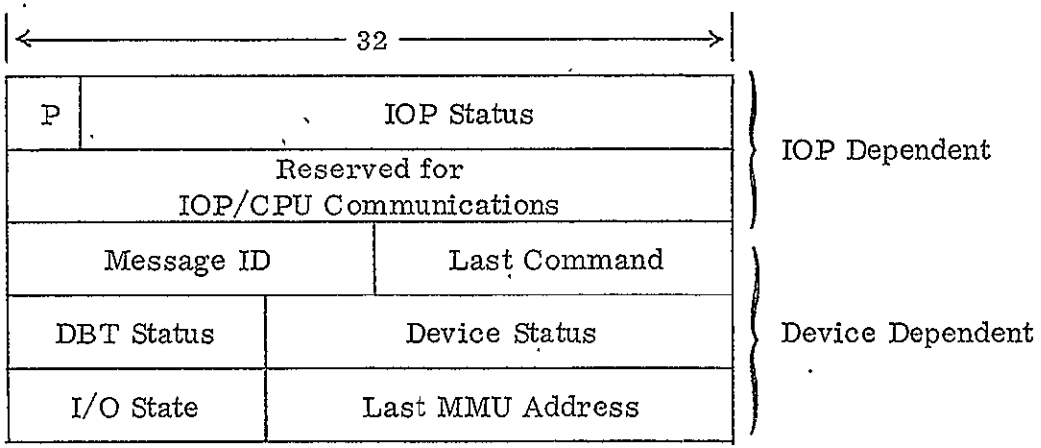
The SCU control word is used to retain the current status of those operations that require a multiple-step CPU/SCU dialog. Its format varies between and within operations.

As noted, SPM is not directly addressable. Specific instructions are provided to access the interrupt status and arithmetic fault mask/status indicators. Portions of the IOP communications packet meaningful to the software executive are provided through the interrupt service entrance mechanism.

(4) Addressing scheme. Figure 28 illustrates an SPM addressing scheme, utilizing the U-flip-flop defined earlier which permits access only to user mode related and non-dedicated sections during user mode operation and all SPM locations during system mode operation. If cell assignments are selected by function, this mechanism (or a similar one) is sufficient to perform all tasks involving scratch pad memory.

3. Special Instructions. In addition to the spectrum of instructions referenced in the discussion of the SUMC baseline, other instructions are outlined here. These additional instructions fall into one of two classes: "required" for MEC operation or "desirable" for additional programming effectiveness. Those which are required have to do with configuration, interrupt, process, input/output, and lockout control, while increased effectiveness is gained by special instructions for recovery and trace, debug execution and system mode control, list and stack manipulation, and program linkage.

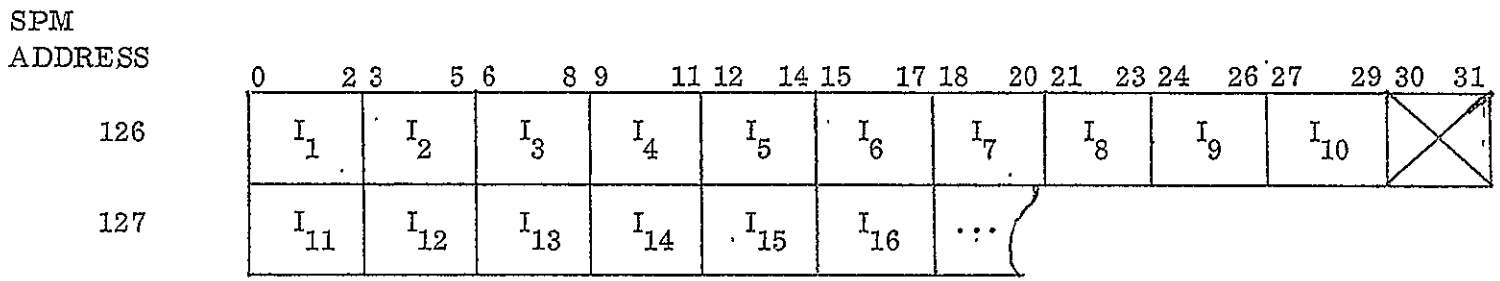
a. Configuration control. A method for configuration control has been outlined conceptually elsewhere /1/ and summarized previously in this report. Slight modifications to the related instructions are incorporated




P - Protect Bit

FIGURE 25

IOP COMMUNICATIONS PACKET

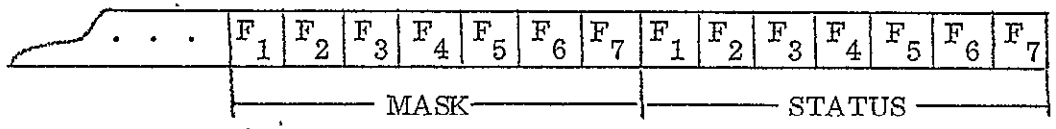



$I_i$ - (i = RELATIVE PRIORITY OF INTERRUPT)

= 000 - DISABLED
= 001 - ENABLED AND DISARMED
= 010 - ARMED
= 011 - QUEUED AND DISARMED
= 100 - QUEUED AND ARMED
= 101 - ACTIVE
= 110 - EXECUTION DISCONTINUED
= 111 - NOT USED

FIGURE 26

INTERRUPT STATUS FORMAT




$F_1$ - ADD/SUBTRACT OVERFLOW

$F_2$ - ADD/SUBTRACT UNDERFLOW

$F_3$ - DIVIDE ERROR

$F_4$ - MULTIPLY OVERFLOW

$F_5$ - FLOATING POINT ERROR

$F_6$ - IMPROPER SQUARE ROOT

$F_7$ - IMPROPER TRIGONOMETRIC FUNCTION

IF MASK BIT $F_i$ = 1 - THE CONDITION DESIGNATED BY "i" WILL GENERATE AN ARITHMETIC FAULT INTERRUPT.

= 0 - THE CONDITION DESIGNATED BY "i" WILL BE IGNORED BY THE INTERRUPT SYSTEM.

FIGURE 27

ARITHMETIC FAULT MASK AND STATUS

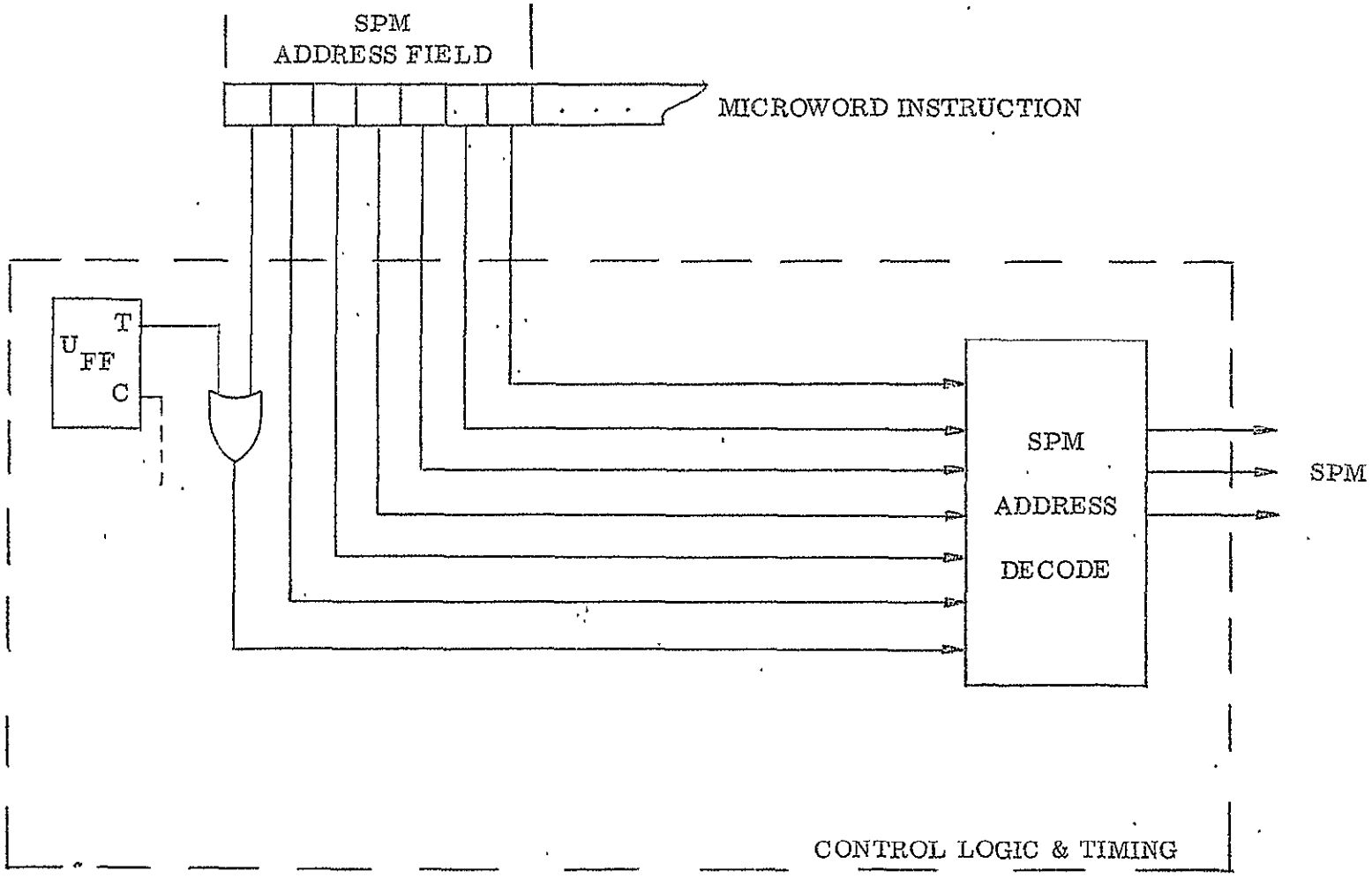



FIGURE 28

SPM ADDRESS GENERATION

into the instructions depicted in table 7. While SCU bus connections to CPUs could have been included, this was not done since the connections can be made in the SCU at switch time with no loss in generality.

The instructions are grouped under four functional headings:

- Connection,
- Disconnection,
- Status Determination, and
- Miscellaneous.

b. Process control. The reader may wish to refer at this point to an earlier discussion during which basic conceptual ideas relating to process control were reviewed. Executive control of processes is facilitated by the definition of a set of primitives. An approach to implementation can be found in an earlier cited report /9/.

Execution of a primitive effects an immediate or subsequent state transition of a process. Table 8 contains the salient points of each primitive instruction.

(1) Start. Execution of a START primitive for a process in the "idle" state results in its immediate transition to the "ready" state. (See table 5 for process state definitions.) Concurrently, the PCB variable "w," known as the work variable, is incremented.

When a process is not in the "idle" state, execution of the START primitive results only in the incrementing of the work variable. The START primitive specifies the object process name as an argument.

(2) Stop. The STOP primitive is invoked by a process to indicate execution completion. The PCB startaddress entry is copied into the returnaddress entry and the work variable, "w," is decremented. If "w" is zero the process returns to the "idle" state. If "w" is not zero it is returned to the "running" state, the next instruction fetch address being in returnaddress of the PCB. The STOP primitive connotes the implied argument, processname, of the invoking process.

(3) Wait. This primitive is executed by a process when in the "running" state and cannot proceed until some arbitrary, requested event has occurred. The process is placed in the "waiting" state until a cooperative process executes the CONTINUE primitive at which time a transition occurs to the "ready" state.

TABLE 7

CONFIGURATION CONTROL (1 of 4)

| CONNECTION INSTRUCTIONS | | | |
|---|---|---|---|
| MNEMONIC CODE | ARGUMENTS | MAP | MEANING |
| PC | CP, MI, MO, II, IO | SM* | Place CPU in SM; make an entry in the SM showing the central processor element in plug position CP connected to MMU buses MI and MO, and IOP buses II and IO. |
| PI | IP, MI, MO | SM | Place IOP in SM; make an entry in the SM showing the input/output processor element in plug position IP connected to MMU buses MI and MO. |
| CM | MM, IA, OA, MI, MO | SM | Connect MMU to buses; make an entry in the SM showing the main memory element in plug position MM connected through its input access port IA and its output access port OA to the main memory buses MI and MO, respectively. |
| CI | IP, IA, OA, II, IO | SM | Connect IOP to buses; make an entry in the SM showing the input/output processor element in plug position IP connected through its input access port IA and its output access port OA to the input/output buses II and IO, respectively. |
| PV | T, V, IC1, IC2, IC3, OC1, OC2, OC3 | SM | Place VDSC in SM; make an entry in the SM showing the VDSC element in plug position V with its input channels 1, 2, and 3 connected to buses IC1, IC2, and IC3, respectively, and its output channels 1, 2, and 3 connected to buses OC1, OC2, and OC3, respectively. The type of VDSC is specified, thus, by T: |

| T | TYPE |
|---|---|
| 000 | VMI |
| 001 | VMO |
| 010 | VII |
| 011 | VIO |
| 100 | VSI |
| 101 | VSO |

* SM - System Map

TABLE 7

CONFIGURATION CONTROL (2 of 4)

| DISCONNECTION INSTRUCTIONS MNEMONIC CODE | ARGUMENTS | MAP | MEANING |
|---|---|---|---|
| SOC | CP, B, IO | AM | Switch-out CPU; switch-out the central processor in plug position CP disconnecting it from all buses (B = 00), the MMU bus only (B = 01), or the IOP bus only (B = 10). IO specifies both input and output (= 00), input only (01), or output only (10). |
| SOM | MM, A, IO | AM | Switch-out MMU; switch-out the main memory in plug position MM disconnecting it at its access port number A (A = 0 implies all ports). IO is interpreted as in the SOC instruction. |
| SOIB | IP, B | AM | Switch-out IOP from buses; switch-out the input/output processor located in plug position IP from all buses (B = 0), the MMU input bus only (B = 01), the MMU output bus only (B = 10), or both MMU buses (B = 11). |
| SOII | IP, A, IO | AM | Switch-out IOP from IOP buses; switch-out the input/output processor located in plug position IP disconnecting it at its access port number A (A = 0 implies all ports). IO is interpreted as in the SOC instruction. |
| SOV | T, V, C | AM | Switch-out VDSC. The VDSC Type T in plug position V is disconnected from its buses as indicated by C:<br>if C = 01, input C1 only,<br>if C = 10, input C2 only,<br>if C = 11, input C3 only,<br>if C = 00, all input and output. |

TABLE 7

CONFIGURATION CONTROL (3 of 4)

| STATUS DETERMINATION INSTRUCTIONS | | | |
|---|---|---|---|
| MNEMONIC CODE | ARGUMENTS | MAP | MEANING |
| SCC | CP | AM | Sense central processor-connect status; if central processor plug position CP is connected to a set of buses, skip the next instruction. (When a processor plug position is vacant, it is assumed that it is disconnected from all buses. The disconnect operation should occur automatically upon manual unplugging or under program control.) |
| SCP | CP | AM | Sense central processor plugged-in status; if central processor plug position CP has an element plugged in, skip the next instruction. |
| SCG | CP | AM | Sense central processor good status; if central processor plug position CP has a good element plugged in, skip the next instruction. |
| SMC | MM | AM | Sense memory connect status; similar to SCC. |
| SMP | MM | AM | Sense memory plugged-in status; similar to SCP. |
| SMG | MM | AM | Sense memory good status; similar to SPG. |
| SIC | IP | AM | Sense input/output processor connect status; similar to SCC. |
| SIP | IP | AM | Sense input/output processor plugged-in status; similar to SCP. |
| SIG | IP | AM | Sense input/output processor good status; similar to SCG. |
| SBG | B, BN | AM | Sense bus-good status; if bus number BN in bus group<br>if B = 000, all bus groups, if B = 100, IO only,<br>if B = 001, MI only, if B = 101, SI only,<br>if B = 010, MO only, if B = 110, SO only.<br>if B = 011, II only,<br>is marked good, skip the next instruction. Note: If B = 00, all buses must be good to cause a skip. |
| LFI | R1 | AM | Load failure indicators. The R, S, and L status indicators for all bus sets are loaded into register R1 for program testing (no specific field format for R1 is assumed at this time). |

TABLE 7

CONFIGURATION CONTROL (4 of 4)

| MISCELLANEOUS INSTRUCTIONS | | | |
|---|---|---|---|
| MNEMONIC CODE | ARGUMENTS | MAP | MEANING |
| CMM | M1, M2 | AM | Connect Memory-Memory; connect memory plug position M1 as indicated in the Action Map for memory plug position M2. |
| CCC | C1, C2 | AM | Connect Central Processor-Central Processor; connect processor plug position C1 as indicated in the Action Map for processor plug position C2. |
| CII | IP1, IP2 | AM | Connect input/output processor-input/output processor; connect the IOP located in plug position IP1 as indicated in the Action Map for IOP in plug position IP2. |
| SWJ | A, BA | AM | Switch and jump; transfer Setup Map information to the SCU switch control logic for switching and save it in the Action Map. Status indicators are set in AM to show associated connections and SM is cleared. Control of all connected processors is transferred simultaneously to memory location A of the memory element whose bank address is BA. |
| SMB | MM, BA | AM | Set the bank address of the memory element located in plug position MM to contain BA. |

TABLE 8

PRIMITIVES (1 of 2)

| PRIMITIVE | ARGUMENTS | DESCRIPTION | PCB ACTION |
|---|---|---|---|
| START | PROCESSNAME | Process is transferred to "ready" state if in "idle" state. Transferred to "ready suspended" state if in "idle suspended" state. | w + 1 → w<br>If (a b c) = (x 0 0);<br>(x 0 1) → (a b c) |
| STOP | PROCESSNAME* | The work variable, "w," is decremented. If > 0 execution is restarted. Otherwise the process is transferred to the "idle" state. | w - 1 → w<br>STARTADDRESS →<br>RETURNADDRESS<br>If w > 0;<br>(0 1 0) → (a b c)<br>If w = 0;<br>(0 0 0) → (a b c) |
| WAIT | PROCESSNAME*<br>$\Delta t$** | The process is transferred to the "waiting" state. | (0 1 1) → (a b c)<br>PC → RETURNADDRESS |
| CONTINUE | PROCESSNAME | Process is transferred from the "waiting" state to the "ready" state. If currently suspended it is transferred to the "ready suspended" state. | (x 0 1) → (a b c) |
| EXIT | PROCESSNAME* | Process is transferred from the "running" state to the "idle" state and subsequently deleted. | (0 0 0) → (a b c)<br>0 → w |
| * Implied<br>** Optional | | PC - Program Counter | |

TABLE 8

PRIMITIVES (2 of 2)

| PRIMITIVE | ARGUMENTS | DESCRIPTION | PCB ACTION |
|---|---|---|---|
| ABORT | PROCESSNAME** | Process is transferred from the "running" state to the "idle" state and subsequently deleted. If suspended, the process is transferred to the "idle suspended" state. | (x 0 0) → (a b c)<br>0 → w |
| SUSPEND | PROCESSNAME | The process is transferred from its current state to its companion suspended state. | (1 x x) → (a b c)<br>PC → RETURNADDRESS |
| RELEASE | PROCESSNAME | The process is transferred from its current suspended state to its companion non-suspended state. | (0 x x) → (a b c)<br>If (a b c) = (0 1 0);<br>RETURNADDRESS<br>→ PC |
| ** Optional | | PC - Program Counter | |

The WAIT primitive may optionally specify a delay, $\Delta t$, the expiration of which will result in the CONTINUE primitive being invoked. The implied argument of the WAIT primitive is the name of the invoking process.

(4) Continue. Execution of the CONTINUE primitive effects a transition of the object process from the "waiting" state to the "ready" state. Subsequent execution of the object process resumes at the instruction sequence following the point at which the WAIT primitive was invoked. The CONTINUE primitive must specify the name of the object process as an execution parameter.

(5) Exit. The EXIT primitive is invoked by a process for the purpose of self-termination. The process is temporarily transferred to a "terminate" state followed by release of its allocated main memory and PCB residence. Again, the implied argument of this primitive is processname.

(6) Abort. This primitive has all the effects of an EXIT and additional capability to perform or request actions to aid in debugging or failure isolation. Execution of an ABORT primitive with an argument, processname, is used by a cooperative process that is aware of an anomaly in the object process. If the processname is not specified, it is implied to be that of the invoking process.

(7) Suspend. Execution of the SUSPEND primitive allows the invoking executive process to effect a non-destructive suspension of an object process. While suspended the object process relinquishes its PCB to be examined or altered dynamically by the invoking process. The state transition of the object process is to companion suspended states for "idle," "ready," "running," and "waiting." The conceptual effects of the START, CONTINUE, and ABORT primitives are preserved during suspension by incrementing "w," transition to "ready suspended," and transition to "idle suspended," respectively.

Suspension is terminated by execution of its converse, the RELEASE primitive. An argument, processname, is required for execution of SUSPEND.

(8) Release. The act of invoking the RELEASE primitive will cause the object process to revert back to its companion non-suspended state. The RELEASE primitive must specify the processname of the object process.

c. Memory access lockout. A main memory unit global data file structure may require that one process prevent access from all other processes for the duration of an operation (an uplink file being compacted, for example). Analogously, an otherwise re-entrant routine may require that a

portion of itself not be executed simultaneously by more than one process. The only effective non-aleatory procedure is memory test and set, TS, utilizing a unique control line to MMU (see earlier discussion of MMU).

TS is a privileged instruction, that may be implemented as an executive request in the user mode. The format is equivalent to a standard memory reference instruction. Operation is as follows:

- The referenced memory location is retrieved and tested by MMU,
- If not all ones, it is set to all ones and the instruction following TS is skipped,
- If all ones, it is restored and the instruction following TS is executed.

A TS condition is cleared by storing zero in the test and set location.

d. Recovery/trace. Table 9 lists three instructions which aid the development of an automatic "checkpoint/restart" procedure executed to recover from transient and intermittent errors. Each instruction in the table enables a program to determine a preceding point of the current instruction sequence.

TABLE 9. RECOVERY/TRACE INSTRUCTIONS

| INSTRUCTION MNEMONIC | OPERANDS | DESCRIPTION |
|---|---|---|
| LCA | R | The address from which the last subprogram call originated is stored in the register designated by R. |
| LJA | R | The address from which the last jump originated (not a subprogram call) is stored in the register designated by R. |
| LPC | R | The address of the instruction executed immediately preceding the current instruction is stored in the register designated by R. |

An additional application exists in the area of error analysis and debugging aids. The following examples illustrate this:

- Under process abort conditions, a "walk-back" listing giving machine conditions at selected points in the instruction path, and

- Determining which of several sequences is entering a given sequence under erroneous conditions.

e. Program and concept verification. Ultimate reliability (and cost) of software can be improved by facilities aiding the programmer during checkout. Among those that relate to hardware/firmware include:

- Breakpoint address - a CPU halt upon encountering a previously specified program address as a result of an instruction fetch;

- Breakpoint operand - a CPU halt upon encountering a previously specified instruction operand (effective address); and

- Data pattern break - a CPU halt upon encountering a previously specified operand value.

Implementation might be in the form of a CPU "debug" mode that can be controlled externally (or internally under program control). An alternative is a separate SUMC model dedicated to software verification.

f. Input/output. Input/output instructions, referred to as Program Controlled Output (PCO), are commands to the IOP to execute an I/O sequence. In order to specify the instruction the following items are necessary:

- Operational specifications,
- IOP designation, and
- Buffer address or pointer to the further information required.

The operational specification or op code designates the particular action required of the IOP. The following actions defined in table 10 have been found desirable:

- Start Input/Output (SO)
- Terminate Input/Output (TO)

TABLE 10

PROGRAMMED CONTROL INSTRUCTION DEFINITION

| MNEMONIC CODE | OPERANDS | DESCRIPTION |
|---|---|---|
| SO | IOP, Class*, Address | The IOP delineated is commanded to perform the I/O operation designated by the Class operand. Class indicates the location of the ECOs (Main Memory or Format Memory). Address designates the first word address (FWA) of the ECO command packet that specifies the operation. |
| TO | IOP, Class, Address | The I/O operation defined by the operands is terminated. Subsequent actions are defined by the command packet. |
| GS | IOP, R, Address | The status indicator variable designated by the address operand is received from the IOP and stored into the SPM register defined by "R." |
| ID | IOP, R, Address | One 32 bit word is sent to the CPU register "R" by IOP, from the device designated in the operand address. |
| OD | IOP, R, Address | One 32 bit word is sent from CPU register "R" by IOP, to the device denoted by the contents of the address operand. |
| DD** | B, X, D, Device Address | The diagnostic process located at effective address is executed upon the device specified by Device Address. |

* Class indicates location of ECOs (MMU or Format Memory)

** DD is a two word instruction; first word contains conventional SUMC Base, Index, Displacement terms for effective address calculation; second word contains device address.

- Get Status (GS)
- Input Direct (ID)
- Output Direct (OD)
- Diagnose Device (DD)

The IOP designation is the select code for the IOP that will execute the operational sequence. Similarly the buffer address or pointer is the main or format memory address of further specification data.

g. List/stack operations. Definition of a set of list/stack operations alleviates the execution overhead for dynamic storage allocation, assists in processing real-time interrupts, and provides convenient, least-redundant methods for implementing re-entrant/recursive routines. Table 11 is a self-explanatory set of list/stack instructions.

Figure 29 depicts a modified ring structure illustrating a viable approach to implementation. The state variable "S," the stack ID, is contained in a main memory table to which there is a pointer in SPM. The format of S depends upon the detailed implementation scheme. Pointers within the list structure are truncated to 16 bits, suggesting stack residence in the lower 65K of memory unless a compensating mechanism is used, for instance, allotting a stack base address. Since the required memory addresses do not conform to normal boundary checking procedures, it is suggested that the list/stack operations be privileged and incorporated as executive requests in the user mode.

h. Interrupt processing. Selection of instructions to facilitate interrupt processing was directed by the criteria that SPM not be directly addressable using the procedure for addressing MMUs, but rather be addressed via micrologic decoding of the instruction op code. Table 12 lists candidate op codes to set and sense associated SPM locations and provide capability for the executive to respond to a priority interrupt. The following candidate instructions have been identified:

- Interrupt Mask Set (IMS),
- Interrupt Arm/Disarm (IAD),
- Clear Interrupt (CLI),
- Reset Interrupts (RIN),
- Set Interrupt Address (SIA),
- Read Interrupt Indicator (RII),
- Set Arithmetic Fault Mask (SAI),
- Enter Interrupt State (EIS), and
- Exit Interrupt State (EFS).

As noted, these instructions are discussed in table 12.

TABLE 11

LIST/STACK INSTRUCTIONS (1 of 3)

| MNEMONIC OP CODE | OPERAND | MEANING |
|---|---|---|
| OS | S, W, L | Open Stack. Define a stack whose address is S (an arbitrary n-bit integer) with a single entry consisting of W words and a maximum length of L entries. Future references to S do not require specification of W and L. |
| CS | S | Close Stack. Destroy the definition and existence of the stack whose address is S. This allows S to be redefined. |
| PU | S, A | Push. Place the W words beginning at A on the top of the stack S. If this new entry will exceed the specified maximum length L ignore the instruction and fetch the next instruction; otherwise skip the next instruction after execution. |
| PO | S, A | Pop. Copy the top entry on stack S into the W words beginning at A and remove the top entry from the stack thus making the next entry the new top. If prior to performing the PO there are no entries on the stack, ignore the instruction and fetch the next instruction; otherwise skip the next instruction after execution. |
| RE | S, A | Read Entry. Copy the top entry on stack S into the W words beginning at A. If prior to performing the RE there are no entries on the stack, ignore the instruction and fetch the next instruction; otherwise skip the next instruction after execution. |
| TP | S | Top. Move a logical pointer to the top entry on stack S. If the stack has no entries, ignore the instruction and fetch the next instruction; otherwise skip the next instruction after execution. |

TABLE 11

LIST/STACK INSTRUCTIONS (2 of 3)

| MNEMONIC OP CODE | OPERAND | MEANING |
|---|---|---|
| RW | S, R, N | Read Word. Copy the contents of the Nth word ($0 \leq N \leq W - 1$) of the stack (S) entry pointed to by the logical pointer (referenced to henceforth as the "logical entry") into register R. |
| WW | S, R, N | Write Word. Replace the contents of the Nth word ($0 \leq N \leq W - 1$) of the stack (S) logical entry by the contents of register R. |
| MD | S | Move Down. Move the logical pointer to the next lower entry on the stack S. If there is no next entry, ignore the instruction and fetch the next instruction; otherwise skip the next instruction after execution. |
| MU | S | Move Up. Move the logical pointer to the next higher entry on the stack S. If no next higher entry exists (pointer is at the top) ignore the instruction and fetch the next instruction; otherwise skip the next instruction after execution. |
| DL | S, R, N | Down Less-than. Beginning with the logical element, scan each succeeding (lower) element until one is found whose Nth word has contents numerically less-than the contents of register R. If the stack is exhausted before the test is satisfied, fetch the next instruction; otherwise, define the satisfying element to be the new logical element and skip the next instruction. |
| DE | S, R, N | Down Equal. Same as DL but test for equivalence of contents. |
| DG | S, R, N | Down Greater Equal. Same as DL but test for word contents greater than or equal to register contents. |

TABLE 11

| LIST/STACK INSTRUCTIONS (3 of 3) | | |
|---|---|---|
| MNEMONIC OP CODES | OPERANDS | MEANING |
| BT | S | Bottom. Position the logical pointer to the last element on the stack. If the stack is empty, fetch next instruction; otherwise skip the next instruction after execution. |
| OT | S | Out. Remove and destroy the logical entry from the stack, closing the two adjacent entries together to reform the stack without changing the relative order of remaining entries. Define the logical entry to be the entry which previously followed the removed logical entry. If no entry previously followed the removed logical entry define the top entry (this also may not exist) on the stack to be the logical entry and fetch the next instruction. Otherwise, skip the next instruction. |
| IN | S, A | In. Insert a copy of the W words beginning at A into the stack as the entry following the logical entry. Redefine the logical entry to be the newly added entry. If the stack is empty prior to the IN instruction; the new entry will be inserted on the stack and defined to be the logical entry. |
| LP | S, R | Logical Pointer. Places address of logical pointer for stack S into register R. |

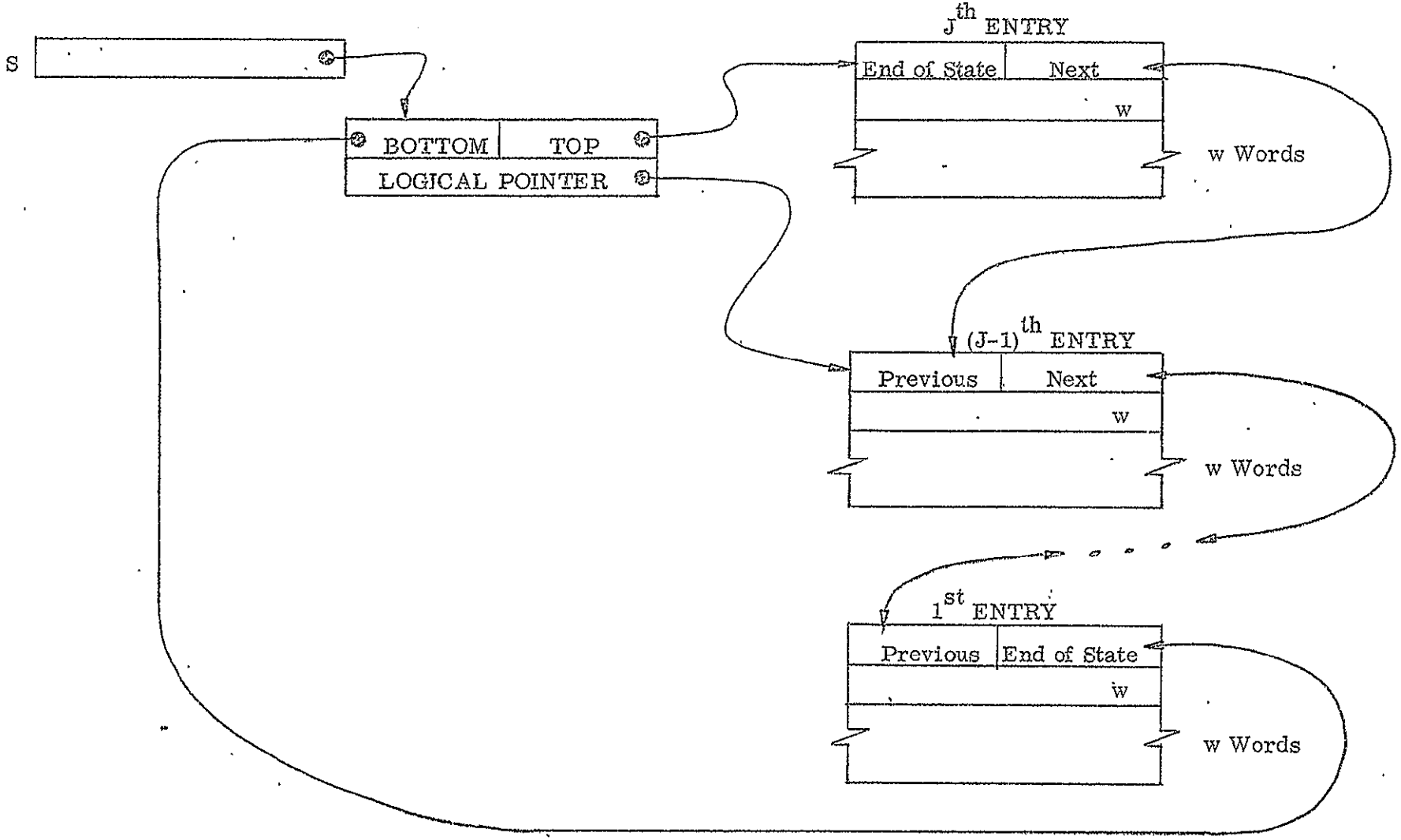



FIGURE 29

STACK IMPLEMENTATION

TABLE 12

INTERRUPT PROCESSING INSTRUCTIONS (1 of 2)

| MNEMONIC OP CODE | OPERANDS | DESCRIPTION |
|---|---|---|
| SIA | R, I | Set Interrupt Address. Store the contents of register R into the SPM interrupt location of which the immediate operand I is the relative priority level. |
| RII | A | Read Interrupt Indicator. Store the contents of the interrupt status and arithmetic fault words (SPM 125 and 126) into locations A and A + 1. |
| SAI | R | Set Arithmetic Fault Mask. Store the contents of register R (bits 25-31) into the mask portion of the arithmetic fault indicator. The initial condition of this register is all ones. A bit set for a particular condition will allow an arithmetic fault interrupt if that condition occurs. |
| EIS | | Enter Interrupt State. A higher priority level interrupt has occurred. Clear interrupt; set Executive mode ff. Branch to process to isolate interrupt source. Consequent action depends on state of system at instant of interrupt arrival. |
| EFS | | Exit from Interrupt. Arm selected interrupts, PCB (Program Counter) – PC, set user mode ff, FETCH. It should be noted that the PCB must contain the appropriate process parameters. |
| IMS | B, I, A | Interrupt Mask Set. Location 2 = B + I + A contains a mask which is used to enable or disable selected interrupt levels where 1 = enabled. The initial state of all interrupts is "disabled." An occurrence of a disabled interrupt is ignored (i.e., null processed). The preceding is accomplished by storing the contents of the effective address in the SPM Interrupt Mask location. |

TABLE 12

INTERRUPT PROCESSING INSTRUCTIONS (2 of 2)

| MNEMONIC OP CODE | OPERANDS | DESCRIPTION |
|---|---|---|
| IAD | B, X, D | Interrupt Arm/Disarm. Location Z = B + X+ D contains a mask which is used to arm or disarm selected interrupt levels. An occurrence of a disarmed interrupt is "remembered" but not serviced until armed. |
| CLI | | Clear Interrupt. The active (highest) interrupt level is reset, allowing queued lower (or equal) interrupt levels to be serviced after execution of the following CPU instruction. |
| RIN | | Reset Interrupts. Pending interrupt service requests are cleared. To obtain service the signal must be reinitiated. |

Instructions IMS and IAD (enable/disable, arm/disarm, respectively) require the following definitions:

- An interrupt level that is enabled will be given CPU service time according to its priority and "armed" status upon each occurrence;
- An interrupt level that is disabled will be ignored;
- An interrupt level that is armed will compete for CPU service time on the basis of priority upon occurrence;
- An interrupt level that is disarmed and requesting service will not be allowed CPU time until armed. In contrast to the disabled state, however, it will be queued for later service.

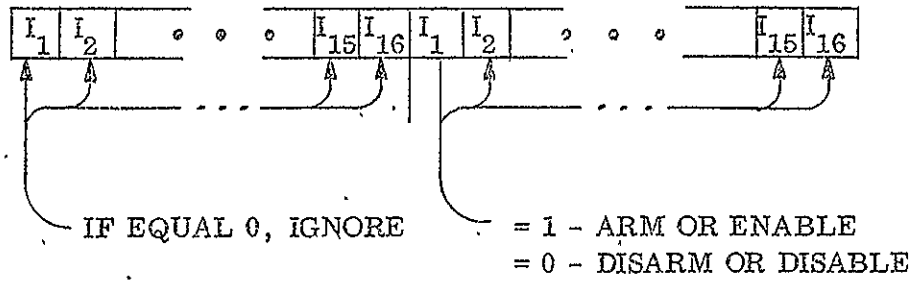
IMS and IAD submit a mask word, as depicted in figure 30, for interpretation. The left half of the mask selects the interrupt levels to be affected and the right half selects the appropriate status redesignation.

Arithmetic fault interrupts may be prevented for specific conditions via an SAI (Set Arithmetic fault Indicator). The format of the required operand mask may be obtained from figure 30.

i. Program linkage. Specification of program linkage instructions (CALL, RETURN, Executive Request) might be influenced by definition of the CPU program structure. Multiple entry points are a simple example of a program structure capable of being facilitated by linkage instructions. More elaborate structures might include programmed "filters," invoked upon entry and exit from a program, which perform parameter checks and set execution conditions. This framework creates a closer functional relationship between program entry and exit instructions. Time requirements for the present study do not permit a thorough evaluation of the program structures required, thus it might be necessary to give complete program linkage specification at a later time.

## C. System Control Unit (SCU)

This section summarizes the role of the system control unit and suggests an approach to its architecture. The SCU acts as a system supervisor at the lowest level at which such control is usually found in the form of an executive routine. That is, the next successive lower level of control is typically found in stored or digital logic. Because the functions allocated to




$I_i$ - RELATIVE INTERRUPT LEVEL

LEFT HALF OF REGISTER IS USED TO SELECT INTERRUPT LEVELS

RIGHT HALF IS USED TO SELECT STATE

FIGURE 30

ENABLE AND ARM MASKS FOR IMS AND IAD INSTRUCTIONS

the SCU comprise an important part of the interface with what is frequently referred to as "hardware," they become prime candidates for microprogram or digital logic implementation.

The functions allocated to the SCU are principally supportive in that events, occurring under program control at one or more of the system's CPUs, initiate action of the SCU. The SCU maintains state information on all of the hardware elements and software elements. The software elements, referred to as processes, are scheduled for allocation of certain hardware resources by the SCU. The methodology and schema by which CPU time is allocated are known collectively as process control. Through process control implicit control of CPU resources is achieved.

In addition to process control, the SCU maintains a map of the physical system, known as the action map (AM), wherein all topological information regarding the connection of elements to other elements within the system is kept. This map serves also as a basis for determining the availability of all elements, including spares. These functions are discussed below in a way that illustrates the relation of the SCU to CPUs.

1. SCU Operations/Functions. Configuration control, CPU control (by way of process control), and process control are summarized as follows.

a. Configuration control. Special instructions to be executed by CPUs under executive control were itemized in a prior section (table 7). With respect to certain of these instructions, requests by the executing CPU are made of the SCU as follows:

(1) Switch and jump request. This request is made as a result of CPU execution of the SWJ instruction. The response of the SCU is to

- Receive and save the jump address parameters from the CPU,
- Receive the setup map (SM) from the CPU,
- Transform the SM into switch commands and switch all affected elements as required,
- Set the AM to reflect SM settings, and
- Send an "Executive Transfer" command to all active CPUs along with the associated jump address parameters.

(2) Disconnect element request. This request is made as part of the CPU execution of an SOC, SOM, SOIB, SOII or SOV instruction. The SCU response is to

- Receive an element switch control word from the CPU,
- Construct associated switch control command(s),
- Issue the switch command(s), and
- Update the AM as required.

A tentative control word format is given in figure 31. The symbols are interpreted as shown in table 13.

TABLE 13. SWITCH CONTROL WORD FIELD DEFINITIONS

| SYMBOL | VALUE | MEANING |
|---|---|---|
| R | 00 | Central Processor (Ignore P) |
| | 01 | Main Memory (P specifies port) |
| | 10 | Input/Output (P specifies port) |
| | 11 | VDSC (Ignore P; C gives TMR channel) |
| P | 0 | All Ports |
| | N | Port N only |
| B | 00 | All Buses |
| | 01 | Processor Input Bus |
| | 10 | Processor Output Bus |
| | 11 | Spare |
| C | 00 | All |
| | 01 | Input C1 |
| | 10 | Input C2 |
| | 11 | Input C3 |
| E | N | Unit Address |

(3) Copy connect request. This request results from CPU execution of a CMM, CCC, or CII instruction. The SCU response is to

- Receive a copy control word from the CPU,
- Search the AM for all element E2 bus connections and save them,

| Type | Port | Bus/ Chan. | Element | Name |
|---|---|---|---|---|
| T | P | B/C | E | Symbol |
| 2 | 3 | 2 | 5 | # Bits |

FIGURE 31

SWITCH CONTROL WORD FIELD FORMAT

- Disconnect element E2,
- Connect element E1 in the same way element E2 was connected, and
- Update the AM appropriately.

Figure 32 shows a tentative copy connect word format wherein T and E1/E2 have the same meaning as the T and E, respectively, of figure 31.

(4) Configuration status request. This request is made by a CPU as a result of having executed an SCC, SCP, SCG, SMC, SMP, SMG, SIC, SIP, SIG, SBG, or LFI instruction. SCU response is to

- Receive the status type code word (format not specified),
- Search the AM to get required status,
- Form an appropriate status response word (format not specified), and
- Send this word to the requesting (waiting) CPU.

Status can be requested for the following Boolean parameters: bus good, central processor connected, memory connected, input/output processor connected, element disconnected, element plugged-in, and element good.

In addition to the responses of the SCU to CPU initiated requests, the SCU initiates commands to a CPU as follows (CPU responses to these commands are detailed elsewhere):

(5) Executive transfer command. This command is issued as the final SCU action in response to a switch and jump request.

(6) Receive VDSC error indicators command. Whenever a VDSC indicates a disagreement in TMR majority voting, this command is issued by the SCU simultaneously to the three active TMR system CPUs.

b. Process control. The role of processes and their control via a set of system primitives defined as SUMC instructions were previously reviewed. It was stated that the functional responsibility for primitive execution was shared by the CPU and the SCU. Three related SCU to CPU commands (preempt, dispatch, increment w) were described by delineating CPU response. Each primitive is assigned a unique command which is transmitted as part of

| Name : | Type | Element 1 | Element 2 |
|---|---|---|---|
| Symbol : | T | E1 | E2 |
| # Bits : | 2 | 5 | 5 |

FIGURE 32

COPY CONNECT WORD FIELD FORMAT

its execution by a CPU to the SCU. An argument, processname, identifies the object process and is received by the SCU following the command. Additional response to the command by the SCU is frequently based on the state of the object process.

(1) Start process request. This command results from the execution by a CPU of a START primitive. The following response is evoked from the SCU.

- The SCU locates (in its memory) the PCB of the object process.
- If the process state is "idle," the w field of the PCB is incremented, the process state bits are set to indicate the "ready" state, an entry is made in the "ready" list, and the dispatcher routine is executed by the SCU.
- If the process state is "ready," "waiting," "ready suspended," "running suspended," or "waiting suspended," the w field of the PCB is incremented.
- If the process state is "idle suspended," the w field of the PCB is incremented and the process state bits are set to indicate the "ready suspended" state.
- If the process state is "running" the w field of the PCB is incremented, the CPU number is identified by fetching it from the PCB, and an "increment w" command is sent to the CPU.

(2) Stop process request. This command originates with the execution by a CPU of a STOP primitive for a process with zero in the PCB w field after decrementing. The following SCU response is executed.

- The PCB is received from the CPU.
- The CPU number field of the PCB is cleared.
- The process state bits are set to indicate the "idle" state and the process' ready list entry is deleted.
- The dispatcher is executed.

(3) Wait process request. Execution of a WAIT primitive by a CPU transmits this command to the SCU. The following SCU response is effected.

- The PCB is received from the CPU.
- The CPU number field of the PCB is cleared.
- The process state bits are set to indicate the "waiting" state and the process' ready list entry is deleted.

(4) Continue process request. This command corresponds to execution of the CONTINUE primitive by a CPU and initiates the following SCU response.

- If the process is in the "waiting" state, the process state bits are set to indicate the "ready" state and an entry is made in the "ready" list; the dispatcher is executed.
- If the process is in the "waiting suspended" state, the process state bits are set to indicate the "ready suspended" state.

(5) Suspend process request. This command is transmitted to the SCU during execution of the SUSPEND primitive by a CPU. The SCU responds in the following manner.

- If the object process is in the "running" state, a "preempt" command is sent to its associated CPU, and the SCU receives the process PCB. The process "ready" list entry is removed. If the object is not in the "running" state, this step is skipped.
- The state bits of the PCB are set to indicate a companion suspended state.
- A copy of the PCB is sent to the CPU executing the SUSPEND primitive.

(6) Release process request. Execution of a RELEASE primitive by a CPU generates this command to the SCU. The response of the SCU is dependent on the state of the object process.

- If the process state is "running suspended," the process state bits are changed to indicate the "running" state, the

identity of the object process CPU is ascertained, and a "dispatch" command issued to the CPU. The process "ready" list entry is inserted and the dispatcher is executed.

- If the process state is "idle suspended," "ready suspended," or "waiting suspended," the PCB state bits are changed to indicate the companion non-suspended state. In the case when the companion state is "ready," a ready list entry is inserted and the dispatcher is executed.

(7) Exit process request. The EXIT primitive, executed for process self-termination, causes this command to be sent to the SCU. The SCU responds in the following manner.

- The PCB is stored in a temporary "terminate" list from which the system allocator may retrieve the CPU number and main memory locations to be released. The process' "ready" list entry is removed.

- The dispatcher is executed.

(8) Abort process request. This command corresponds to execution of the ABORT primitive and initiates the following SCU response.

- If the process is in the "running" state, a "preempt" command is sent to the object process CPU and the PCB is received.

- The object process PCB is stored in a temporary "terminate" list from which the system allocator may retrieve the CPU number and main memory locations to be released. Additionally, error analysis procedures may be initiated from the allocator.

- The process' "ready" list entry is removed and the dispatcher is executed.

c. CPU control. Control of system CPUs by the SCU is accomplished through the collective actions outlined above. These are summarized here to emphasize the total impact of the SCU.

(1) CPU availability. The SCU makes CPU elements available as a computational resource through the accomplishment of switching actions that bring an element on-line to a specific configuration.

(2) CPU resource allocation. Through the complementary actions of dispatching and preempting CPUs according to an SCU executed allocation algorithm, computational time is distributed among all competing program processes.

(3) CPU replacement. By accomplishing the determination of an element's topographical connections with other system elements, the SCU can effect replacement with similar spares.

(4) CPU failure detection and isolation. Through the ability to structure internally redundant system subsets and recognize notification of disagreements with a majority, the SCU can separate transient and apparent hard failures and initiate corrective action.

2. SCU Architecture. The functional nature of the SCU must be examined in greater detail before an optimal architecture can be specified. Very little arithmetic capability seems necessary. This is indicated by the predominantly logical nature of the tasks having to do with process and configuration control. The dispatching function may or may not require arithmetic capabilities depending on the details of the associated process selection algorithm.

On the other hand, if the MEC specification is to be general enough to satisfy a broad spectrum of mission profiles, it is clear that a high degree of flexibility in the form of open-endedness is desirable. A rather open-ended approach results from postulating that the SCU will be implemented with a general purpose computer such as the SUMC. As a baseline, the SUMC represents a strong departure point and is therefore assumed in this report.

In order to provide a tangible implementation basis, the functional nature of the SCU is discussed below in some detail by way of emphasizing peculiarities. In particular, the action map is rough drawn, a summary of functions is given, basic instructions are specified, and a list of SUMC-oriented specifications comprising an implementation framework are provided. Table 14 gives a summary of the SCU characteristics derived from these considerations.

a. Functional overview. Figure 33 depicts a functional diagram of the SCU. The major data structures, located in local memory, are seen to be the action map (AM), process control blocks (PCBs), and ready list (RL). Inputs to the control and timing function from a clock, CPUs, or TMR VDSCs cause activation of the Process Control, Configuration Control, or Faults functions. The functions Insert, Remove, Dispatcher, Exit, Abort and Faults can be invoked by the Process Control function and have to do with control of software processes and allocation of CPU time. The Configuration Control

TABLE 14

SCU SUMMARY

| | |
|---|---|
| LOCAL MEMORY: | |
| Word Size (bits) | 18 |
| Cycle Time (nanoseconds) | 500-1000 |
| Number of Words | 8192 |
| SUMC/SCU: | |
| Data/Register Path Width | 16 |
| Word Size (bits) of SPM | 16 |
| Cycle Time (nanoseconds) of SPM | 50 |
| Number of Words in SPM | 8 |
| Number of Words in IAROM | 64 |
| Word Size (bits) of IAROM | 9 |
| Bit Width of Seq. Cntrl. Unit | 9 |
| Number of Words in MROM | 512 |
| Word Size (bits) of MROM | 50 |

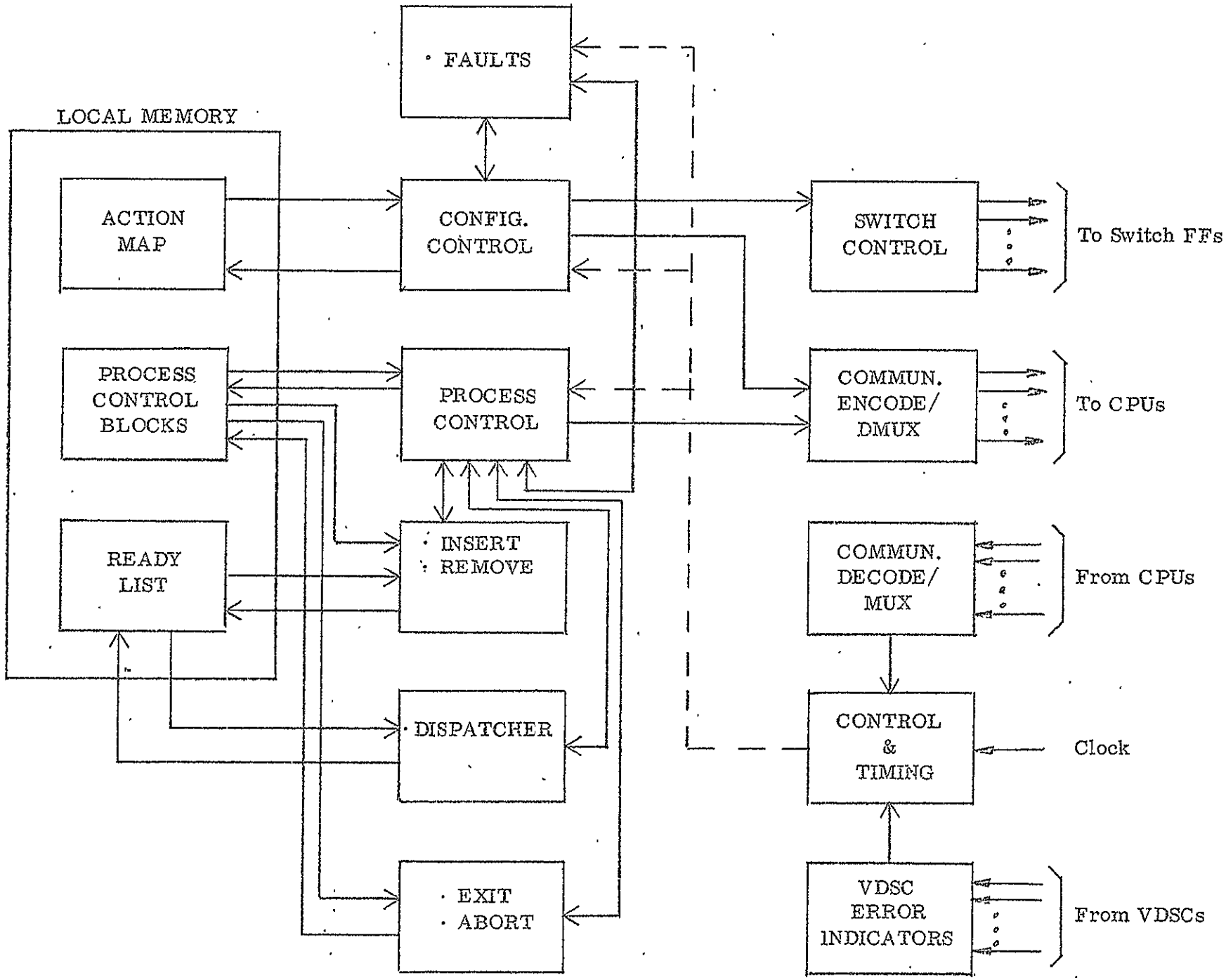



FIGURE 33

SCU FUNCTIONAL DIAGRAM

function, which handles switching of system elements, can be invoked either by Faults or a CPU under program control. A summary of SCU functions includes:

- Configuration Mapping,
- Configuration Switching,
- Configuration Status Reporting,
- Process Dispatching (CPU Allocation),
- Process State Error Analysis and Recovery,
- Process State Transition Monitoring,
- Ready List (Job Stack) Manipulation,
- Adaptive Process Control, and
- Adaptive Configuration Control.

b. Action map structure. To support an estimate of LM size, this section describes a feasible structure for the action map. Figure 34 shows how a record is maintained showing which processor plug position is connected to each MI and MO bus. In addition, the M and AP fields designate one memory plug position and its connected access port. A main memory unit connector block, as shown in figure 35, is used to record the connection (to a given bus) of additional MMUs. Each (M, AP) pair serves to specify a unique connector block and the unique location within the block of another (M, AP) pair connected to the same bus. Thus, the designator pairs form a chain linking together all MMUs connected to a particular bus. Two connector blocks (one each for MI and MO connections) are required for each MMU. The MMU is depicted as having twelve (12) access ports, one for each of eight (8) CPUs and four (4) IOPs.

Figure 36 contains bus connection data for CPU/IOP and CPU/SCU; connection is via the II/IO and SI/SO buses, respectively. The (I, IP) pairs serve to link together multiple IOPs connected to a common bus. The scheme is identical to that discussed above for MMUs and requires two (2) connector blocks, as shown in figure 37, for each IOP.

Finally, figure 38 shows a possible data structure for recording and maintenance of VDSC connections when system operation is redundant.

The system action map is seen to require somewhat under thirty-two (32) words (based on a 32 bit word) not including connector blocks. Assuming thirty-two (32) MMUs and four (4) IOPs, and packing the blocks, approximately two hundred eighty (280) additional words are required. Although status indicators (not shown) must also be maintained for all elements, these can be packed into the unused space of the action map. Therefore, the total (32 bit wide) space required for the AM is three hundred twelve (312) words. A sixteen (16) bit wide space of six hundred twenty-four (624) is adequate.

| | ← PROCESSOR/MEMORY → | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | MI | | | | | MO | | | | |
| B | U | P | M | AP | US | U | P | M | AP | US |
| 0 | | | | | | | | | | |
| 1 | | | | | | | | | | |
| 2 | | | | | | | | | | |
| 3 | | | | | | | | | | |
| 4 | | | | | | | | | | |
| 5 | | | | | | | | | | |
| 6 | | | | | | | | | | |
| 7 | | | | | | | | | | |
| 10 | | | | | | | | | | |
| 11 | | | | | | | | | | |
| 12 | | | | | | | | | | |
| 13 | | | | | | | | | | |
| # Bits: | 2 | 3 | 5 | 4 | 2 | 2 | 3 | 5 | 4 | 2 |

LEGEND

| | | | |
|---|---|---|---|
| B | Bus Number | P | Processor Position # |
| U | Usage | M | Memory Position # |
| | = 00 Not Used (Good Spare) | AP | Access Port # |
| | = 01 P is CPU | MI | Memory Input Bus |
| | = 10 P is IOP | MO | Memory Output Bus |
| | = 11 Not Good | US | Unused Space |

FIGURE 34

ACTION MAP CONNECTIONS FOR PROCESSOR/MMU

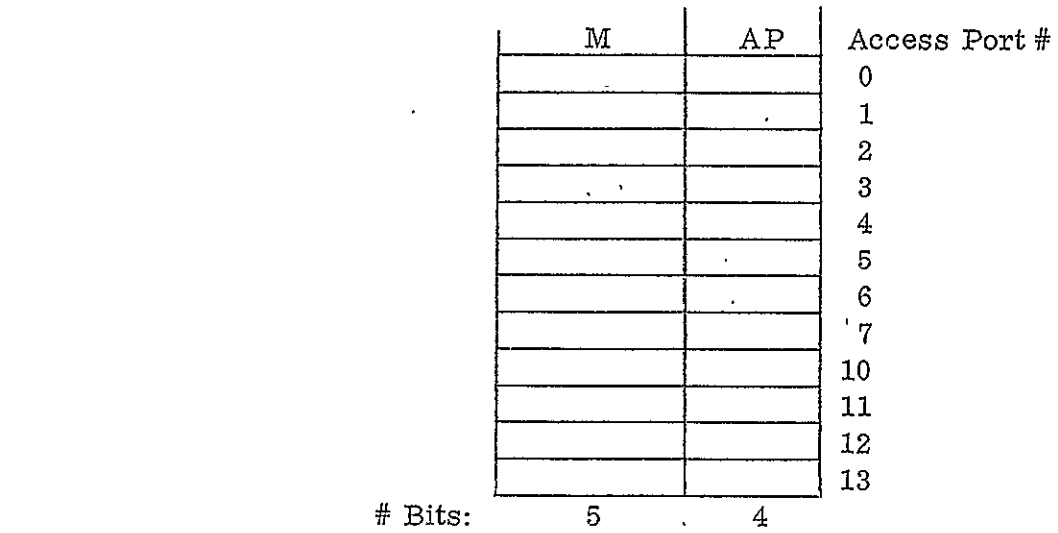



FIGURE 35

MMU CONNECTOR BLOCK

| | CPU/IOP | | | | | | | | CPU/SCU | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | II | | | | IO | | | | SI | | | SO | | |
| B | G | C | I | IP | G | C | I | IP | G | C | SP | G | C | SP |
| 0 | | | | | | | | | | | | | | |
| 1 | | | | | | | | | | | | | | |
| 2 | | | | | | | | | | | | | | |
| 3 | | | | | | | | | | | | | | |
| 4 | | | | | | | | | | | | | | |
| 5 | | | | | | | | | | | | | | |
| 6 | | | | | | | | | | | | | | |
| 7 | | | | | | | | | | | | | | |
| # Bits: | 1 | 3 | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 3 | 1 | 3 | 3 |

LEGEND

| | | | |
|---|---|---|---|
| G | Good Indicator | IP | IOP Access Port # |
| | = 0 Good | II | IOP Input Bus |
| | = 1 Not Good | IO | IOP Output Bus |
| C | CPU Position # | SI | SCU Input Bus |
| I | IOP Position # | SO | SCU Output Bus |

FIGURE 36

ACTION MAP CONNECTIONS FOR CPU/IOP AND CPU/SCU

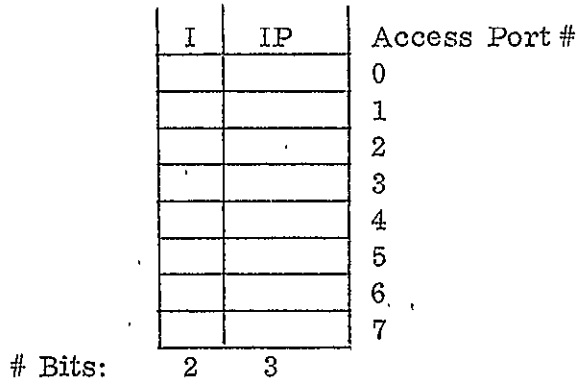



FIGURE 37

IOP CONNECTOR BLOCK

| | ← PROC/MEM → | | | | ← CPU/IOP → | | | | ← CPU/SCU → | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | MI | | MO | | II | | IO | | SI | | SO | | |
| B | VMI | VC | VMO | VC | VII | VC | VIO | VC | VSI | VC | VSO | VC | US |
| 0 | | | | | | | | | | | | | |
| 1 | | | | | | | | | | | | | |
| 2 | | | | | | | | | | | | | |
| 3 | | | | | | | | | | | | | |
| 4 | | | | | | | | | | | | | |
| 5 | | | | | | | | | | | | | |
| 6 | | | | | | | | | | | | | |
| 7 | | | | | | | | | | | | | |
| 10 | | | | | | | | | | | | | |
| 11 | | | | | | | | | | | | | |
| 12 | | | | | | | | | | | | | |
| 13 | | | | | | | | | | | | | |
| # Bits: | 3 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 2 |

LEGEND

| | | | |
|---|---|---|---|
| VMI | VDSC MI Bus | VSO | VDSC SO Bus |
| VMO | VDSC MO Bus | VC | VDSC Channel |
| VII | VDSC II Bus | | = 00 Not Connected |
| VIO | VDSC IO Bus | | = 01 |
| VSI | VDSC SI Bus | | = 10 TMR Channel |
| | | | = 11 |

FIGURE 38

ACTION MAP CONNECTIONS FOR VDSCs

c. PCBs and ready list. In addition to the AM discussed above, a process control block (PCB) is required in the SCU's local memory for each MEC system process. Referring to figures 19 and 23, it is seen that each PCB takes twenty-five (25) 32 bit words (including one word to link them together). The determination of the number of software processes requires a knowledge of mission requirements.

Figure 39 depicts a flexible structure for the ready list (RL) where n integral priority levels are accommodated. Each entry is comprised of components for maintenance of a ring at each priority level, a sub-priority indicator, and a pointer to the associated PCB. This ready list structure can accommodate a relatively sophisticated dispatcher and may be considered, at three (3) 32 bit words per entry, to be a liberal structure. The average number of entries in the list is a random variable that is dependent on the mission and cannot, therefore, be estimated without suitable simulation or queueing analysis.

d. Local memory. Based on the above discussion, an expression for the size of a 32 bit LM is available as

$LM_{32}$ = $312 + 25p + n + 3e + s/2$, where
p = number of software processes,
n = number of priority levels,
e = number of entries in RL, and
s = number of software instructions.

As an example, assume p = 30, n = 10, e = 10, and s = 1000. Thus,

$LM_{32}$ = 1602 (assuming 16 bit instructions), and
$LM_{16}$ = 3204.

From this admittedly crude analysis it appears that provisions for an 8K 16 bit LM are necessary for reasonable mission spectrum coverage.

e. Scratch pad memory (SPM). Because of the relatively small LM word size compared to the number of words required, SPM will be required for indexing to access the data structures discussed above. Thirteen (13) bit words would be adequate for this purpose but, since a 16 bit SPM would allow LM and temporary data storage also, this greater width is preferred. See figure 40 for a layout of the eight word SPM.

f. Instructions. The formats shown in figure 41 are recommended for software instructions. These formats were derived from the basic instruction repertoire given in table 15.

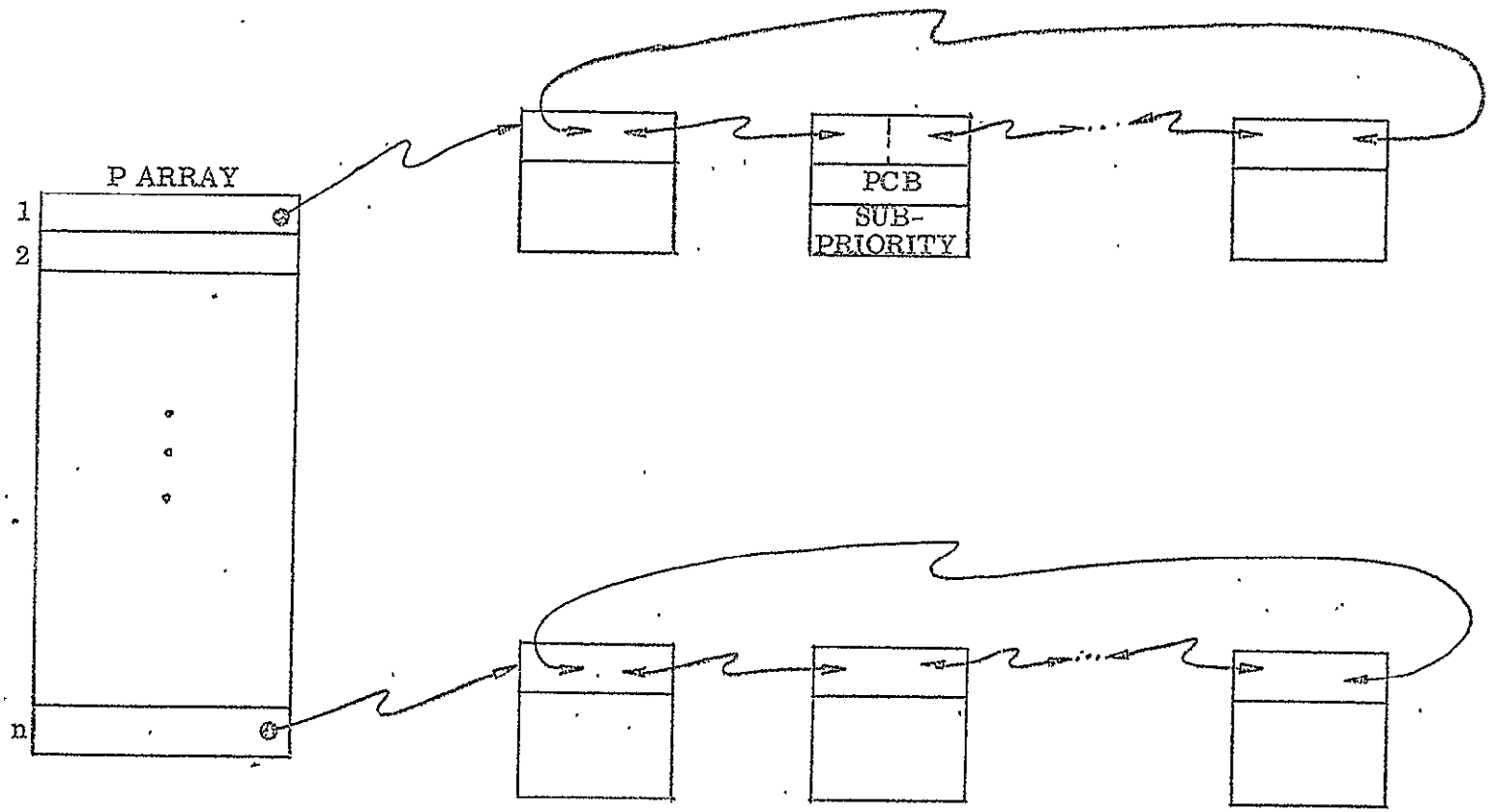



FIGURE 39

READY LIST STRUCTURE

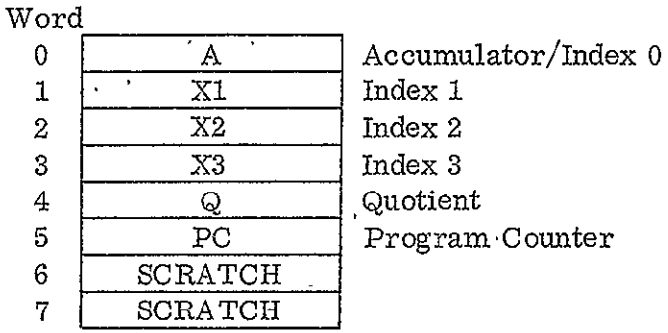



FIGURE 40

SCU SCRATCH PAD MEMORY

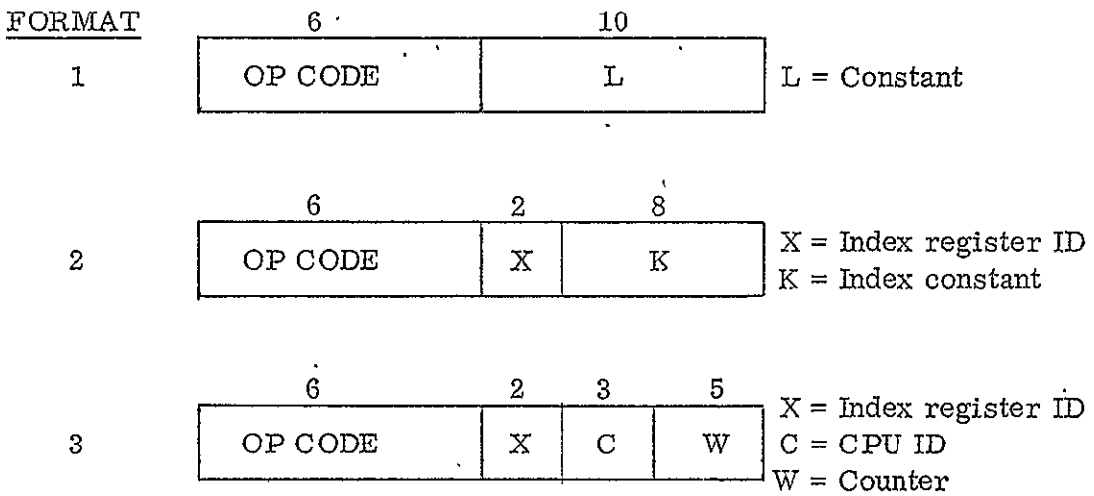



FIGURE 41

SCU INSTRUCTION FORMATS

TABLE 15. SCU BASIC INSTRUCTIONS

| OPERATION | ARGUMENTS | FORMAT | NOTATION | MEANING |
|---|---|---|---|---|
| ENA | L | 1 | $L \rightarrow A$ | Enter A |
| SAR | L | 1 | $(A)_R \rightarrow L$ | Store rightmost (10 bits of) A |
| ENQ | L | 1 | $L \rightarrow Q$ | Enter Q |
| SQR | L | 1 | $(Q)_R \rightarrow L$ | Store rightmost Q |
| J | L | 1 | $L \rightarrow PC$ | Jump |
| JR | L | 1 | $(PC) \rightarrow L$; $L+1 \rightarrow PC$ | Jump Return |
| JI | L | 1 | $(L) \rightarrow PC$ | Jump Indirect |
| ENX | X, K | 2 | $K \rightarrow X$ | Enter Index |
| XA | X | 2 | $(X) \rightarrow A$ | Index to A |
| AX | X | 2 | $(A) \rightarrow X$ | A to Index |
| SAQ | | 1 | $(A) \rightleftarrows Q$; $(Q) \rightarrow A$ | Swap A and Q |
| IA | L | 1 | $(A)+L \rightarrow A$ | Increase A |
| IX | X, K | 2 | $(X)+K \rightarrow X$ | Increase X |
| L | X, K | 2 | $((X)+K) \rightarrow A$ | Load A |
| S | X, K | 2 | $(A) \rightarrow (X)+K$ | Store A |
| AD | X, K | 2 | $((X)+K) + (A) \rightarrow A$ | Add to A |
| SU | X, K | 2 | $(A) - ((X)+K) \rightarrow A$ | Subtract from A |
| M | X, K | 2 | $(A) \times ((X)+K) \rightarrow AQ$ | Multiply |
| D | X, K | 2 | $(AQ) \div ((X)+K) \rightarrow A$; $R \rightarrow Q$ | Divide |
| OR | X | 2 | $(X) \vee (A) \rightarrow A$ | Or X with A |
| AND | X | 2 | $(X) \wedge (A) \rightarrow A$ | And X with A |
| EOR | X | 2 | $(X) \oplus (A) \rightarrow A$ | Exclusive - Or X with A |
| ML | X, K | 2 | $((X)+K) \wedge (Q) \rightarrow A$ | Masked Load |
| SRA | W | 3 | $(A) \times 2^{-W} \rightarrow A$ | Shift Right A end off |
| RA | W | 3 | $(A) \times 2^{W} + (A) \times 2^{-W} \rightarrow A$ | Rotate A Left (end around) |
| SLAQ | L | 1 | $(AQ) \times 2^{L} \rightarrow AQ$ | Shift Left AQ end off |
| RQ | W | 3 | $(Q) \times 2^{W} + (Q) \times 2^{-W} \rightarrow Q$ | Rotate Q Left (end around) |
| SAE | L | 1 | If (A) = L, Skip | Skip A equal |
| SXZ | X, K | 2 | If (X) = K, Skip | Skip X equal |
| OC | C, X, W | 3 | Output W words beginning at (X) to CPU C | Out to CPU |
| IC | C, X, W | 3 | Input W words beginning at (X) from CPU C | In from CPU |
| DC | C | 3 | Dispatch CPU a | Dispatch CPU |
| PC | C | 3 | Preempt CPU C | Preempt CPU |
| SC | L | 1 | Signal CPU with Command L | Signal CPU |

## D. Input/Output Processor (IOP)

The IOP frees the CPU from the procedures required to accomplish data transfers. The significance of IOPs is shown in figure 1, Multi-Element Configuration.

The IOPs, then, initiate and monitor the following categories of data transfers upon command of a CPU:

- CPU-peripheral device,
- Peripheral device-main memory, and
- Peripheral device-peripheral device.

In the above a peripheral device is considered to be a device that is connected to a DBT. This section outlines IOP operations, instructions, and an architecture for the IOP.

1. I/O Operations. Since the IOP is dedicated solely to this purpose it is germane to permit operation on a polling basis. This will be a cooperative venture between the IOP and the attached DBTs. Trade studies are required to determine whether or not variable polling sequences are required. However, whatever the sequence, the IOP will interrogate each DBT in turn for an I/O demand or response. The device will respond either with an ACK, Reject, or by transmitting the buffered message. Upon successful receipt of the message, the IOP will in turn reply to allow the device to clear its buffer and accumulate further data.

a. Data transfers. For CPU-peripheral device bidirectional transfer, which is considered to be a transfer using the Input Data Direct (ID) and Output Data Direct (OD) instructions to communicate with SPM, the following events must occur:

- The IOP detects the request from the CPU by interpretation of the transmitted data.
- If the request is an OD, the IOP initializes the peripheral device by transmitting an appropriately encoded command. The IOP then raises the POLL REQUEST signal to the CPU. The CPU, after recognition and service of the signal, outputs the first word from its PRR to the access port of the IOP. The IOP performs the appropriate reformatting and outputs the encoded message to the peripheral device which acknowledges receipt. If the CPU command requires notification of a termination condition, the IOP inputs status from the DBT and then signals either successful completion or a failure.

- If the request is an ID, the IOP inputs the information from the device DBT buffer, assembles the word and signals the CPU. After the CPU has polled the IOP, the data is made available on the access port for the appropriate CPU, which can then input the word and signal successful completion or failure to the IOP.

For communication between peripheral devices and main memory, the following factors must be accounted for:

- Upon detection of the request, the IOP activates the appropriate data path between memory and the peripheral device.

- For a read to memory operation the DBT device control buffer is loaded with the device transfer address, the number of words to transfer, and the destination address. Upon command of the IOP, the device begins transmission. The IOP receives each word, and executes the required formatting operations. When the word is assembled, the IOP requests a memory access. When the memory word is written, the IOP indicates ready for the next word. Termination can be invoked either from the device or from the IOP.

- For an output from memory to a device, the approximate converse of the above operations takes place after initialization of the DBT/device control buffers. The IOP requests access to memory and upon receipt of the word, changes the data to data bus format and outputs to the bus. Upon receipt of transmission, the DBT performs any required reformatting of the information into device-amenable format, and transfers the buffered data to the device. Each transmission is acknowledged. Error and/or completion notification signals to the CPU are generated upon termination by the IOP.

- To initiate peripheral device to peripheral device transfer the IOP commands a destination device to be ready to accept data over a defined data path, and acknowledge completion of the transfer. The source device is then commanded to transfer a prescribed number of words.

b. Bus operations monitoring. The IOP must provide for bus operations monitoring of message and hardware status. Message statistics are accumulated to provide a measure of system loading based on queue length.

In a similar way hardware status and activity monitoring must be performed to permit possible reconfiguration in the event of device failure.

These status queues must be available to the CPU in a terse but intelligible format. Status information is transmitted from the IOP to the CPU upon demand. This collated information is based on raw status information transmitted by the DBT to the IOP upon:

- Demand,
- Termination due to completion,
- Termination due to failure, and
- Termination upon command of the CPU.

2. IOP Architecture. An overview of the IOP resulting from the allocation of functions among the different hierarchies of subsystems involved in data transfer is shown in figure 42, Overall Block Diagram of SUMC Implemented as an I/O Processor (IOP).

Although minimum modifications to the basic SUMC, including an additional memory access port for format memory, are required to implement the IOP, optimal performance will require restructuring of the micrologic in the MROM and the addition of appropriate control and status lines between the SUMC and the I/O multiplexers. While Adder 2 is apparently not required for the IOP, a substantial redesign of the SUMC is required to eliminate it, which is beyond the scope of this report.

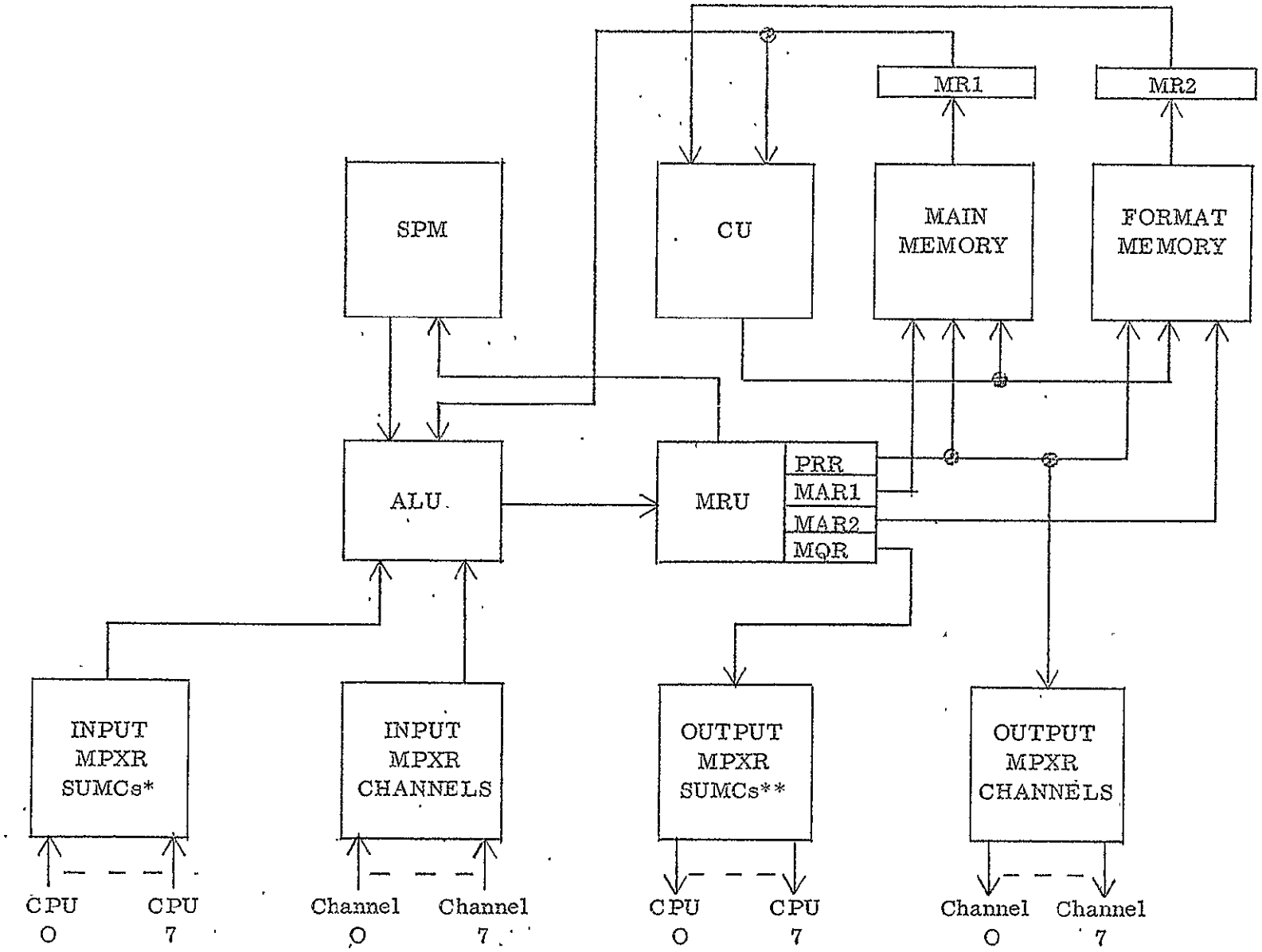
In order to delineate the IOP capability required, the following topics must be analysed:

- IOP to CPU communication,
- IOP program commands, and
- IOP Scratch Pad layout.

a. IOP to CPU communication. As illustrated in figure 43, IOP State Diagram for CPU-IOP Dialog, the following considerations are relevant:

- The dormant or OFF state is exited by applying power to the IOP, sending it to the IDLE state.

- Transition to the READY state is effected by a CPU initialization signal.




* From PRR (16-31) of each CPU
** To MPXB1 (16-31) of each CPU

FIGURE 42

OVERALL BLOCK DIAGRAM OF SUMC IMPLEMENTED AS AN IOP

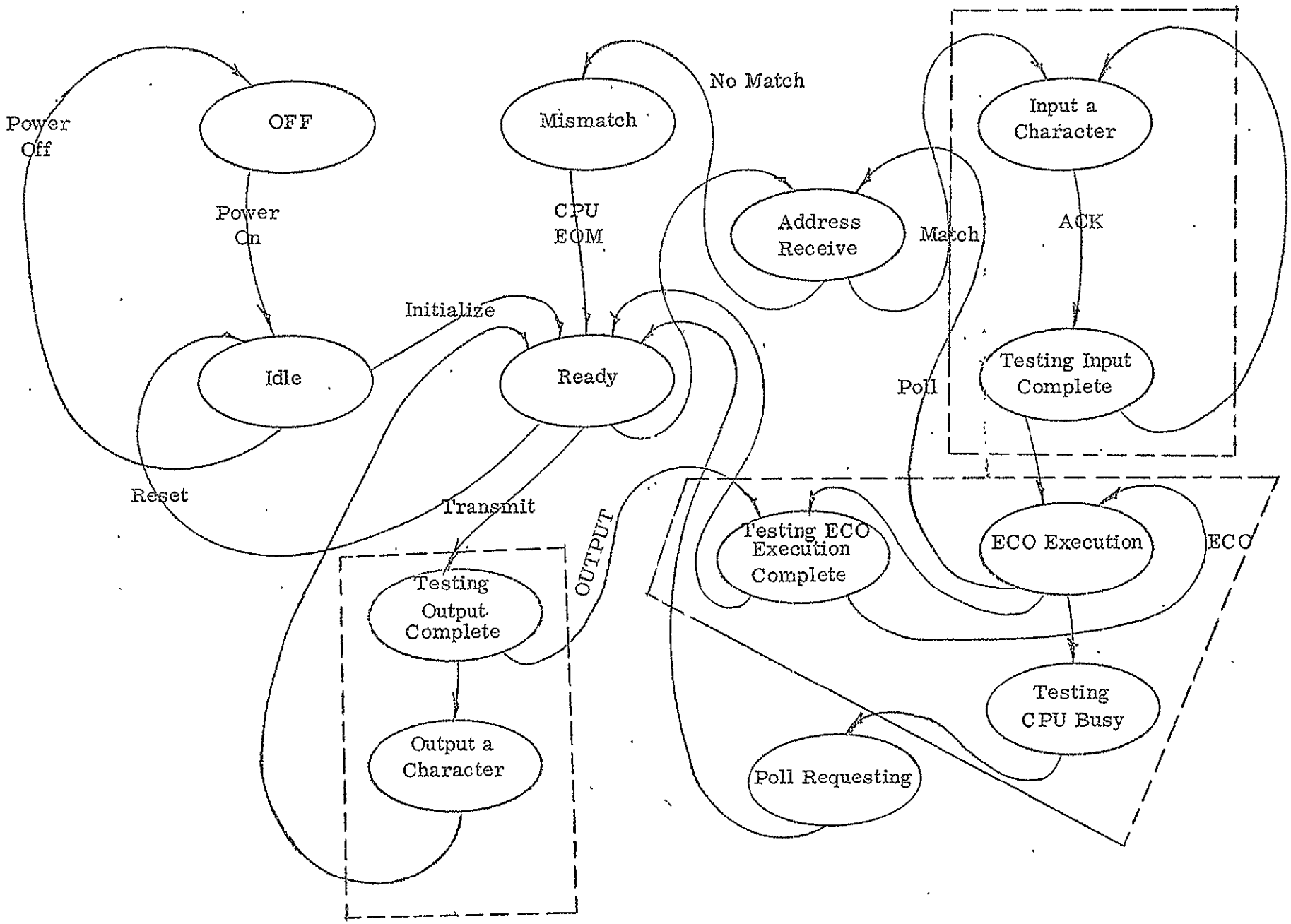



FIGURE 43

IOP STATE DIAGRAM FOR CPU-IOP DIALOG

- If the CPU desires to transmit information to the IOP, it utilizes the Poll signal. This causes the IOP to enter the ADDRESS RECEIVE state. Receipt of an address from the CPU will cause the IOP to compare it with its own ID. Here a match will cause the IOP to enter the INPUT state and transmission may proceed. Conversely a mismatch implies that the CPU has a dialog with a different IOP. Therefore the CPU BUSY marker for the appropriate CPU must be set.

- If the IOP must transmit to the CPU, the POLL REQUESTING state is entered, after the CPU becomes available, causing the Poll Request signal to be output and a transition to the READY state invoked. The CPU ACK signal will cause the OUTPUT state to be convoked, in which information transfer to the CPU can proceed.

- The CPU EOM signal must reset the CPU BUSY signal in each IOP to indicate CPU available.

- The IOP may enter the ECO state from either the INPUT or OUTPUT states.

- Successful higher state terminations lead to the READY state for further CPU commands.

- Anomalies evidenced by a Reject signal cause transition to a higher state ERROR. This state is presently not defined.

Signals utilized in the preceding discussion are shown in figure 21, CPU Control - Bus Communication Output Parameters.

b. IOP program commands (ECOs). In a similar manner to PCOs, additional ECOs are recommended to expand capability of baseline /8/. Recommended ECOs are shown in table 16 as follows:

- WRITE (Output to destination device) (WO),

- READ (Input to destination device) (RI),

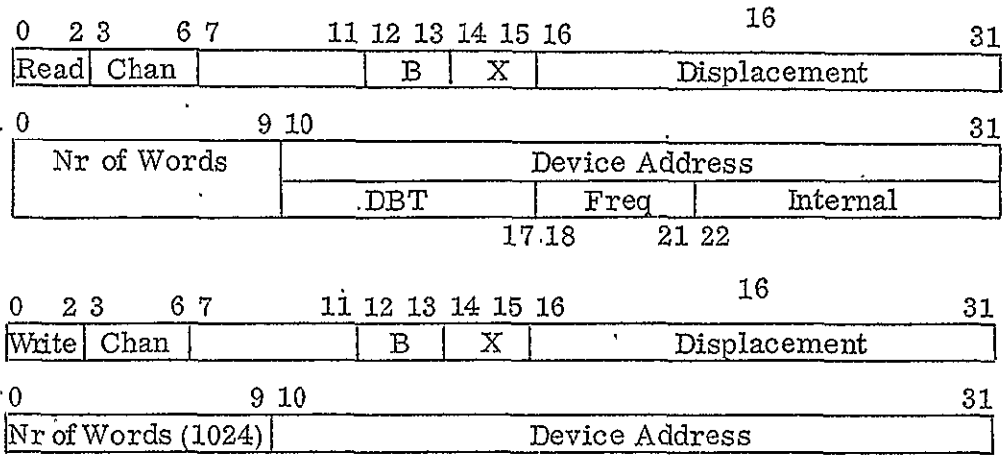
- TRANSFER (Source to destination device) (XF),

- TRANSFER IN COMMAND (Jump) (JU),

TABLE 16. EXTERNALLY CONTROLLED OUTPUT INSTRUCTIONS (ECO)

| MNEMONIC CODE | OPERANDS | DESCRIPTION |
|---|---|---|
| WO | Command Packet (figure 44) | The data path designated by the Command Packet is established. Data transfer from the source to the destination device is initiated. |
| RI | Command Packet | The denoted data path is established. Data transfer from destination device to source device is initiated. |
| XF | Command Packet | A data path is established from source device to destination device. The source device is commanded to transfer the number of words denoted in the Command Packet. |
| JU | Address | Contents of Address — PC, FETCH ECO. |
| HA | - | IOP enters idle state. |
| ACK | CPUID | An acknowledge signal is sent to the denoted CPU. |
| RJ | CPUID | An error signal is sent to the CPU. |
| PO | CPUID | A poll request signal is sent to the designated CPU. |
| LS | R, N, Address | N words of format buffer memory are loaded into SPM beginning at location R, if N > k the instruction is skipped and the succeeding instruction executed, if N k the instruction is executed and the succeeding instruction skipped where k will be derived by future study. |

MMU-Peripheral




Peripheral-Peripheral

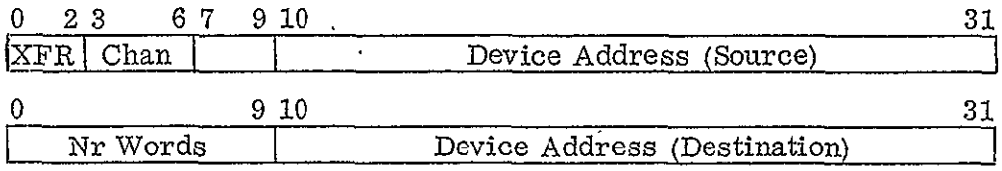



FIGURE 44

ECO COMMAND PACKETS

- HALT (Halt) (HA),
- ACKNOWLEDGE (ACK),
- REJECT (RJ),
- POLL REQUEST (PO),
- LOAD SCRATCHPAD (SPM) FROM MEMORY (LS)

and, in addition to the ECOs,

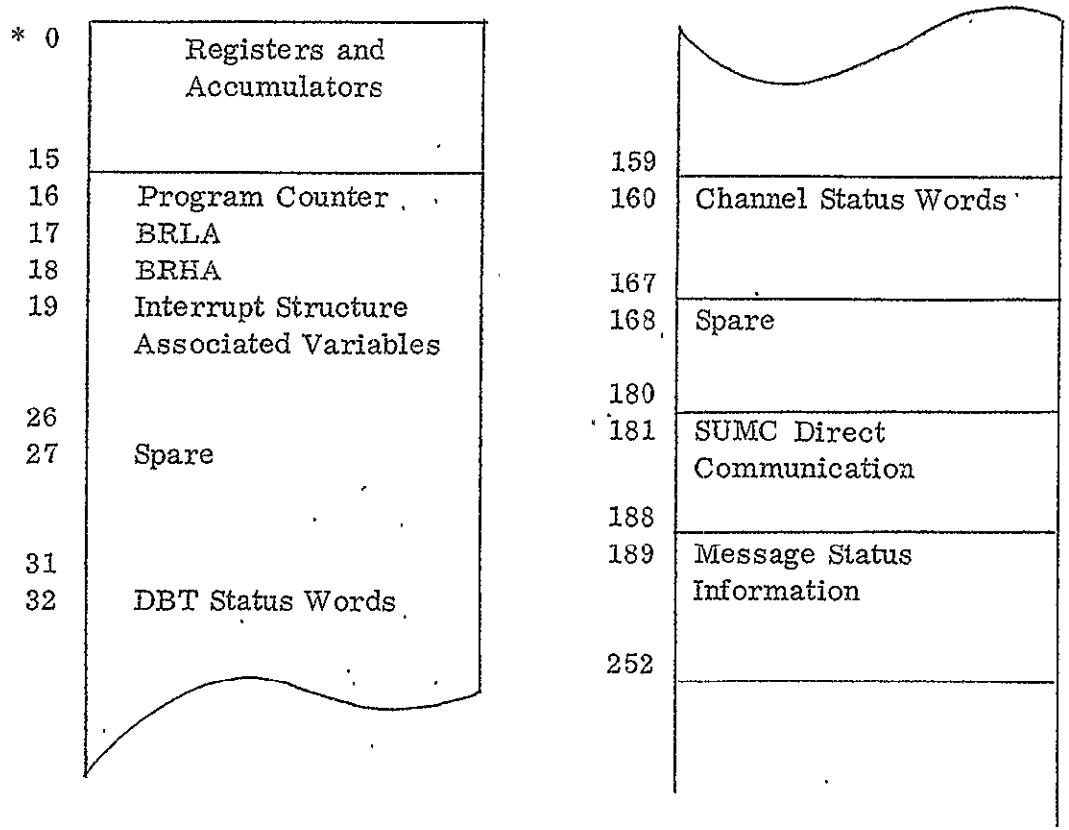
- COMMAND FORMAT (Kernel for output to data bus).

c. IOP scratch pad layout. Based on the preceding discussions, a storage requirements analysis for parameter storage is presented in figure 45, IOP Scratch Pad Memory Configuration.

Parameter storage falls into three areas:

- Variables required to support IOP operations; for example, Program Counter, Index Registers, etc.
- Variables concomitant with maintenance of hardware status information, and
- Variables associated with message status and SUMC-IOP-peripheral device communications.

Presently 256 words of SPM for an IOP appear adequate.




* Note address shown as example only.

FIGURE 45

IOP SCRATCH PAD MEMORY CONFIGURATION FOR PARAMETER STORAGE

APPROVAL

SPACEBORNE COMPUTER MULTI-ELEMENT
SYSTEM CONFIGURATION
ARCHITECTURE REFINEMENT: TASK 1 REPORT

By

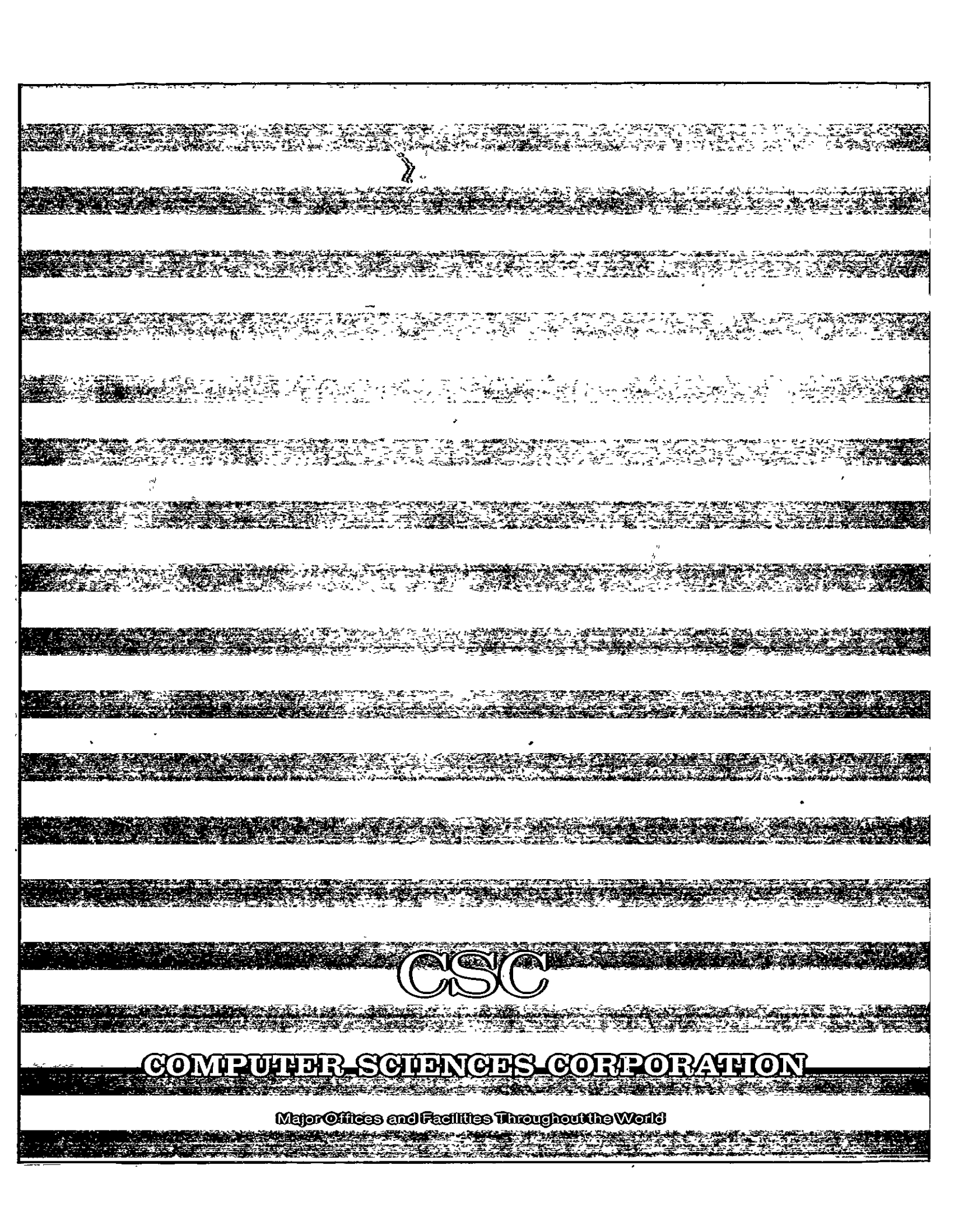
J. R. Kennedy
R. T. Curran
B. P. Buckles
W. A. Hornfeck

The information in this report has been reviewed for security classification. Review of any information concerning Department of Defense or Atomic Energy Commission programs has been made by the MSFC Security Classification Officer. This report, in its entirety, has been determined to be unclassified.

This document has also been reviewed and approved for technical accuracy.

_____

Dr. H. Hoelzer
Director, S&E-COMP


CSC
COMPUTER SCIENCES CORPORATION
Major Offices and Facilities Throughout the World